Universidad
Carlos III de Madrid

UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNICA DE TELECOMUNICACIONES ESPECIALIDAD TELEMÁTICA

PROYECTO FIN DE CARRERA

GLUCOMIDLET
APLICACIÓN MÓVIL ACCESIBLE DE TELEMEDICINA PARA PERSONAS CON DISCAPACIDAD VISUAL

Autor: Javier Rodríguez Rodríguez.

Tutora: Dra. Celeste Campo Vázquez.

Agradecimientos

Este apartado debería ser con toda seguridad el más extenso de esta memoria, ya que después de tanto tiempo, son muchas las personas a las que debo estar agradecido. Esta carrera de fondo comenzó hace ocho años, aunque a todos los efectos parezca que comenzó hace cuatro.

Hoy en día, si a alguien se le pregunta si cree que puedo ser ingeniero, se reiría y diría que sí, ¿por qué no? No obstante, esta pregunta no tenía una respuesta tan clara hace ocho años, y por eso mis primeros agradecimientos son para las primeras personas que confiaron en mí hace mucho tiempo, gracias a Dori, mi profesora de braille del Centro de Recursos para Invidentes del Gobierno Vasco (CRI) y gracias a Amparo, quien se desvivió por mí en mis primeros pasos.

Sin embargo, esta carrera podría haber sido mucho más larga y difícil si no fuese por las personas que he encontrado en la Universidad Carlos III de Madrid; sin ellas no podría haber aprendido a hacer nada de lo que hoy en día sé hacer. Por eso tengo que estar infinitamente agradecido a todos mis compañeros y compañeras de clase porque ellos han hecho que mereciese la pena ir a clase todos los días, a veces de sol a sol.

Gracias al servicio de voluntarios y voluntarias que estuvieron ahí cuando les necesité.

Gracias a todos los becarios y becarias por su increíble esfuerzo, sin el cual, no podría haber aprobado ni una sola asignatura.

Al profesorado de la Escuela Politécnica Superior le tengo que estar agradecido por completo, desde el primer al último profesor o profesora, desde las primeras profesoras que me dieron la bienvenida hasta las últimas que continuaron haciendo un magnífico trabajo. No obstante, hay unos nombres propios a los que les tengo que estar especialmente agradecido.

Tengo que agradecer a Carlos García Rubio el haber confiado en mí para trabajar en la Cátedra Nokia.

A mi tutora Celeste le tengo que agradecer el haber querido dirigir un proyecto sobre el que me ha dejado total libertad y el haber tenido tanta paciencia y ganas de trabajar conmigo, y el apoyo que me ha dado y las veces que tan claras me ha dejado las ideas.

Como no, tengo que agradecer a Alberto García todos estos años de apoyo y de mediación con los profesores, la orientación siempre correcta sobre mi carrera, y el tiempo incontable que me ha dedicado.

Y a Blanca Leiva, gracias por todo el tipo de apoyo sin excepción que me ha dado, que en muchas ocasiones ha trascendido el ámbito académico, por los consejos, las charlas con un café por delante…

Sin dos personas como Blanca y Alberto no podría haber llegado donde he llegado.

Quiero agradecer a mis amigos el haber hecho de mis años en Madrid unos de los más felices.

Y por último, a quien tengo que estar agradecido más que a nadie es a mis padres y a mi novia por haberme dado un apoyo incondicional desde el principio, y especialmente a mi madre, sin la cual en los primeros años no podría haber conseguido nada, y a mi novia Ezkioga sin la cual no podría haber hecho nada hasta el mismo día de hoy.

Esto es para quienes me han enseñado todo lo que sé.

Nadie me enseñó qué significa imposible.

Resumen

En este proyecto se realizará un prototipo de una aplicación de telemedicina para teléfonos móviles en lenguaje Java ME y se hará una especial incidencia en la accesibilidad para personas con discapacidad visual tanto de las partes que conforman la propia aplicación como de las herramientas del lenguaje utilizado para su implementación.

Se analizará la accesibilidad de la aplicación móvil de acuerdo con el uso de lectores de pantalla, la accesibilidad del interfaz web, y la idoneidad de contar con dispositivos que interactúen con un teléfono móvil.

Los dispositivos empleados para la creación del prototipo son un teléfono móvil Nokia 6210 Navigator y un medidor de glucosa Compact Plus de Laboratorios Roche que facilita los niveles de glucosa mediante pitidos.

El prototipo está formado por tres aplicaciones: aplicación móvil, aplicación servidora y aplicación web. El funcionamiento en conjunto de estas tres aplicaciones ofrece la posibilidad a una persona ciega o deficiente visual hacerse una medición de glucosa con un "glucómetro" y posteriormente almacenar dicha medición en su teléfono móvil transmitiendo los datos vía bluetooth desde un PC, haciendo uso de la aplicación servidora, para posteriormente y desde el teléfono móvil, enviarlos vía http a un servidor web encargado de almacenarlos y mostrarlos.

Índice General

Índice de Tablas

Índice de Imágenes

Capítulo 1. Introducción

En este proyecto se realizará un estudio sobre la posibilidad de desarrollar aplicaciones accesibles en lenguaje Java ME. Se implementará un prototipo de aplicación de telemedicina destinado a almacenar y organizar en un teléfono móvil los controles de glucosa realizados por el usuario para enviarlos posteriormente a un servidor web que se encargará de mostrar dichos controles en una página web pensada para ser accedida por el doctor del usuario.

1.1. Motivación del Proyecto

La tecnología es sin duda una poderosa herramienta. Pero, ¿para qué?

Desde un punto de vista comercial, se puede ver a la tecnología como una herramienta para crear necesidades, para hacer que los usuarios desarrollen una dependencia de los productos tecnológicos.

Desde el punto de vista de la ingeniería, la tecnología se puede entender como una herramienta para dar solución a necesidades ya existentes.

En cualquier caso, tanto si el objetivo es crear nuevas necesidades como si es darles una solución, se provoca que el usuario desarrolle dependencia de la tecnología, de modo que no es fácil asegurar por tanto que hay personas dependientes y otras que no lo son, puesto que todos dependemos en mayor o menor medida de algo o de alguien.

En este punto, la tecnología puede jugar un papel importante; puede ser una herramienta de integración o una herramienta de exclusión. En efecto, el uso de la tecnología como mecanismo de integración y el empeño por demostrar que la ingeniería está para solucionar problemas es la principal motivación de este proyecto.

Contando con la motivación principal y teniendo en cuenta la situación del proyectando, deficiente visual, la parte quizás más complicada que es la detección del problema y haciendo de la necesidad una virtud, ha resultado ser la más sencilla. Las dificultades que se encuentra una persona ciega o deficiente visual en su vida cotidiana son familiares al proyectando como lo son a otras muchas personas.

Ha habido también un factor importante que ha motivado este proyecto que es la búsqueda de una solución no solamente accesible, sino usable. Este factor es lo que ha motivado que se pensase desde un principio en desarrollar una aplicación para teléfonos móviles, ya que el "móvil" es un elemento sin el que actualmente no sabemos vivir, y puede resultar por tanto un elemento fuertemente integrador.

Para una persona con discapacidad el acceso a la tecnología se presenta como una cuestión fundamental para desarrollar tareas que a priori pueden resultar sencillas a simple vista, pero que su ejecución se complica según el tipo de discapacidad que tenga la persona. La solución de tareas cotidianas utilizando la tecnología es un campo muy estudiado, pero sin embargo, existen remedios que se centran más en la superación del problema que en cómo accede la persona con discapacidad a esa solución. Por ejemplo, existen aparatos identificadores de colores que hablan el color a una persona ciega. Se supera un problema, pero se crea otro, que es la excesiva dependencia de innumerables

aparatos cuya funcionalidad podría integrarse de manera relativamente sencilla mediante una aplicación software. En la usabilidad de las soluciones es donde el teléfono móvil se presenta como el aparato cotidiano por excelencia, y tal y como se ha comentado, la capacidad que un teléfono móvil tiene para integrar aplicaciones software es un factor clave en el desarrollo de soluciones usables destinadas a realizar actividades diarias; una persona con discapacidad no necesita un aparato identificador de colores, otro lector de códigos de barras, y un reproductor mp3 parlante por ejemplo, sino un teléfono móvil con tres aplicaciones diferentes. Si bien el poder realizar tareas relativamente sencillas usando un teléfono móvil es un trabajo interesante, conseguir que una persona con discapacidad mejore su calidad de vida en cuestiones vitales se presenta como un reto.

Afrontar un reto así es lo que ha llevado a desarrollar, de entre tantas aplicaciones móviles posibles, una aplicación móvil que se centre en facilitar cuestiones relacionadas con la salud del usuario. En concreto, y teniendo en cuenta que se ha pensado en una aplicación accesible para personas ciegas o deficientes visuales, se decidió desarrollar un prototipo destinado a facilitar el control de la diabetes a personas con esta discapacidad. El motivo por el que se ha elegido desarrollar una aplicación para mejorar el control de la diabetes y no de otra enfermedad es que la diabetes está entre las tres enfermedades que mayor número de personas ciegas provoca en el mundo. La diabetes y la ceguera están íntimamente relacionadas, de modo que resulta especialmente interesante desarrollar una aplicación que permita mejorar la calidad de vida a las personas ciegas y diabéticas.

1.2. Objetivos

De manera general, el objetivo principal que se ha marcado en este proyecto es el de crear una aplicación móvil accesible utilizando Java ME. muy ligado a él, está el objetivo de que la aplicación móvil, además de ser accesible para un lector de pantalla, sea también sencilla de usar.

Aunque no se ha definido como objetivo principal, sí que en todo momento se ha querido que el prototipo desarrollado pueda tener en un futuro una aplicación práctica, de modo que a pesar de ser un prototipo, se ha pensado en desarrollar una aplicación cuya funcionalidad pudiese solucionar ciertos problemas que las personas diabéticas y con deficiencia visual pueden tener en su día a día.

Para conseguir el objetivo principal, se han tenido que abordar los siguientes objetivos:

1. Estudiar y decidir qué tecnología móvil es la más adecuada para desarrollar una aplicación web accesible, Symbian o Java ME.
2. Detectar un problema al que se enfrenta una persona ciega o deficiente visual en su vida diaria, e implementar una aplicación que aporte una solución al mismo.
3. Comprobar que la aplicación es efectivamente accesible y robusta frente al manejo común del teléfono móvil.
4. Elaborar la memoria y explicar los conceptos que se han tenido que abordar para realizar el proyecto.

1.3. Contenido de la memoria

Esta memoria se divide en los siguientes capítulos:

- En el capítulo *Estado del Arte* se hablará sobre la accesibilidad del interfaz de alto nivel de Java ME y de sus componentes. Se hará una breve exposición sobre el API bluetooth de Java ME, el JSR-82, así como del API Bluetooth de J2E, BlueCove. También se darán unos trazos sobre qué es la síntesis de voz, qué tipos de sintetizadores de voz hay y cuál es la diferencia entre sintetizador de voz y lector de pantalla. En este capítulo también se podrá leer qué es un medidor de glucosa y qué tipos hay, así como una pequeña descripción acerca de qué es la Telemedicina.
- En el capítulo *Descripción de la aplicación* se hablará sobre el funcionamiento general del prototipo y de las aplicaciones que lo forman, aplicación móvil, aplicación servidora y aplicación web, así como de los terminales soportados.
- En el capítulo *Pruebas* se hará un estudio sobre los resultados obtenidos de las pruebas realizadas, así como de la accesibilidad de los entornos de desarrollo empleados. Se probará la accesibilidad de la aplicación móvil en el terminal correspondiente, la sencillez de su interfaz de usuario así como que la navegación sea intuitiva o no.
- En el capítulo *Historia del proyecto* se dará un repaso al trascurso del proyecto, la implementación del prototipo, las pruebas, las configuraciones, y los trabajos previos de investigación y documentación.
- En el capítulo *Conclusiones y Trabajos Futuros* se repasarán los objetivos logrados pero también aquellas cosas que son objetivamente mejorables en el prototipo. También habrá una sección sobre trabajos futuros donde se propondrán diferentes aplicaciones para desarrollar.
- En el apéndice *Manual de instalación* se explicarán las herramientas necesarias para poder programar las aplicaciones y para poder ejecutarlas. También se expondrá qué configuraciones son necesarias para poder ejecutar el prototipo.

Capítulo 2. Estado del Arte

En este capítulo se hablará sobre la tecnología que se ha empleado para desarrollar el prototipo y que es Java ME, explicando aquellos aspectos que han resultado más relevantes en la implementación de las tres aplicaciones. Como se utiliza tecnología bluetooth para la transferencia de datos, se verá en qué consiste esta tecnología y qué aporta Java para poder utilizarla. También se abordará qué es un medidor de glucosa y qué tipos hay, se verá en qué consiste la síntesis de voz y cuál es su diferencia con un lector de pantalla, y por último, se hará una pequeña introducción a la telemedicina.

2.1. Java ME – Java Micro Edition

La plataforma Java ME [1] es un subconjunto de la plataforma Java, que proporciona una serie de API's (*Application Programming Interfaces*) para desarrollar software destinado a dispositivos con recursos limitados como pueden ser teléfonos móviles, GPS, electrodomésticos, máquinas expendedoras…

En Java, las especificaciones formales y tecnologías propuestas para que sean añadidas a la plataforma son descritas por unos documentos llamados JSR (*Java Specification Request*). Las revisiones públicas formales de JSRs son controladas antes de que los JSR se conviertan en final y sean votados por el Comité Ejecutivo JCP.

Un JSR final suministra una implementación de referencia que da una implementación libre de la tecnología en código fuente y un Kit de compatibilidad de tecnología para verificar la especificación de la API.

La arquitectura de Java ME [2] está basada en los siguientes elementos: configuración, perfil y paquetes opcionales. A continuación se van a desarrollar cada uno de ellos.

Configuraciones de Java ME

Las configuraciones están compuestas por una máquina virtual y por un conjunto mínimo de bibliotecas, y son las encargadas de proporcionar la funcionalidad básica para un conjunto de dispositivos que comparten características similares, como podrían ser la gestión de memoria o el acceso a la red.

Actualmente existen dos tipos de configuraciones que son CDC (Conected Device Configuration) y CLDC (Conected Limited Device Configuration) [3], las cuales están orientadas a diferentes tipos de dispositivos.

CLDC es la configuración orientada a ser usada por dispositivos con recursos más limitados que CDC. En concreto, está pensada para dispositivos con conexiones de red intermitentes, procesadores lentos y memoria limitada, como son los teléfonos móviles. Normalmente, los dispositivos que cuentan con la configuración CLDC, tienen una CPU de 16 ó 32 bits, y una memoria disponible para la implementación de la plataforma Java y sus aplicaciones de entre 128 y 256 Kilobits.

CLDC está basada en la máquina virtual CLDC HotSpot VM.

Por otra parte, CDC es la configuración pensada para dispositivos que cuentan con más memoria, procesadores más potentes y mayor ancho de banda que los dispositivos que cuentan con CLDC. Los dispositivos típicos que incorporan CDC son set-top boxes, internet TV,... CDC incluye una máquina virtual completa y un subconjunto de API's de la arquitectura J2E mucho mayor que la máquina virtual CLDC HotSpot VM.

Los dispositivos que incorporan CDC como configuración cuentan con CPU de 32 bits y un mínimo de memoria disponible de 2 Megabits para la plataforma Java y sus aplicaciones asociadas.

Perfiles en Java ME

Para tener un entorno de ejecución completo orientado a una categoría determinada de dispositivos hay que combinar las configuraciones con un conjunto de API's llamadas Perfiles.

En la actualidad Java ME tiene los siguientes perfiles:

- Mobile Information Device Profile (MIDP) [4].
- Foundation Profile.
- Personal Profile.
- Personal Basis Profile

De los cuatro perfiles anteriores, solamente MIDP es el perfil que está asociado a la configuración CLDC, y por tanto el perfil que será implementado por los teléfonos móviles. MIDP ofrece funcionalidades básicas para teléfonos móviles como la interfaz de usuario, la conectividad a redes, el almacenamiento persistente y la gestión del ciclo de vida. Existen tres versiones de MIDP que son MIDP 1.0 (JSR 37), MIDP 2.0 (JSR 118) y MIDP 3.0 (JSR 271).

- MIDP 1.0 especifica una serie de requisitos hardware como que el dispositivo debe contar con una pantalla de 96x54 píxeles, debe contar con dispositivo de entrada de datos como teclado de teléfono estándar (teclado de una mano), teclado QWERTY (teclado de dos manos) o pantalla táctil, y con una memoria dedicada en exclusiva a aplicaciones MIDP de 128 KBytes de memoria no volátil y 32 KBytes de memoria volátil. Los dispositivos que incorporen MIDP 1.0 también se caracterizarán por tener un ancho de banda limitado bidireccional, inalámbrico y con acceso intermitente a la red.

  Este perfil tiene una serie de limitaciones como son que no tiene soporte de acceso directo a píxeles de las imágenes (RGB Data), no soporta el trabajo a pantalla completa, no cuenta con soporte directo de audio sin una API adicional, solamente soporta http para conexión con el exterior y no puede consultar el estado de las teclas en cualquier momento.

  Los paquetes que incorpora MIDP 1.0 son:

  `javax.microedition.io` (interconexión a redes).
  `javax.microedition.lcdui` (interfaz de usuario).
  `javax.microedition.midlet` (ciclo de vida de las aplicaciones).
  `javax.microedition.rms` (almacenamiento persistente).

- MIDP 2.0 por su parte, requiere una serie de características adicionales con respecto a MIDP 1.0 en los dispositivos que lo quieran ejecutar como son contar con una memoria no volátil de 256 KBytes para la aplicación MIDP, 8 KBytes de memoria no volátil para el almacenamiento persistente y 128 KBytes de memoria volátil para el entorno de ejecución. Con respecto a los requisitos de conexión, MIDP 2.0 establece que tiene que haber conexión a redes bidireccionales con acceso inalámbrico intermitente y con ancho de banda limitado. El dispositivo tiene que ser capaz también de reproducir sonidos.

  MIDP 2.0 introduce una serie de mejoras con respecto a MIDP 1.0 como son permisos y firma de código de aplicaciones, mejoras en la seguridad de operaciones en red.

  Incorporación de capacidades multimedia, interfaces de usuario mejoradas, cadenas de conexión estandarizadas para acceso por puerto serie, cadenas de conexión estandarizadas para datagramas, sockets y sockets de servidor, registro de solicitudes (push registry) que permite que se ejecuten MIDlets en respuesta a conexiones de red entrantes, los repositorios de registros se pueden compartir entre MIDlets e incorpora soporte para juegos.

  MIDP 2.0 incorpora cuatro nuevos paquetes:

  `javax.microedition.lcdui.game` (soporte para juegos).
  `javax.microedition.pki` (seguridad).
  `javax.microedition.media` (soporte multimedia).
  `javax.microedition.media.control` (soporte multimedia).

  MIDP 2.0 también incorpora mejoras en el interfaz de usuario incluyendo un nuevo interfaz y dos nuevas clases: El interfaz `ItemCommandListener` se utiliza para recibir notificaciones de comandos que han sido invocados a través de un objeto `Item`. Las dos clases que incorpora son `CustomItem` que permite crear ítems personalizados y la clase `Spacer` utilizada para definir espacios en blanco y ajustar mejor la interfaz.

- El perfil MIDP más reciente es MIDP 3.0 [5] aprobado en Diciembre de 2009, y presenta otros requisitos de sistema: pantalla de 176x220 pixeles y con una resolución de 16 bits; sistemas de entrada de audio convencionales como son teclados QWERTY, teclado estándar de teléfono, pantalla táctil; 1 MByte de memoria no volátil para la implementación MIDP y 512 KBytes de memoria no volátil para los datos persistentes de las aplicaciones; 1 MByte de memoria volátil para la ejecución de las aplicaciones Java; y soporte para conexiones de red inalámbricas bidireccionales con ancho de banda limitado y ocasionalmente intermitentes.

  Este perfil es una especificación para los dispositivos móviles de tercera generación que amplía todas las funcionalidades relacionadas con la interoperabilidad entre los dispositivos y se basa en el perfil MIDP 2.0 mejorándolo en los siguientes aspectos:

  - Permite la concurrencia múltiple de MIDlets.

- Especifica funcionalidades de firewall, mejora el comportamiento en tiempo de ejecución y cuestiones relacionadas con el ciclo de vida de los MIDlets.
- Permite la ejecución de MIDlets sin interfaz de usuario en background.
- Permite MIDlets autoejecutables, es decir, MIDlets que se ejecutan cuando se enciende el dispositivo.
- Permite comunicaciones entre MIDlets.
- Permite la compartición de librerías entre MIDlets para mejorar la interoperabilidad entre dispositivos.
- Introduce el concepto de Liblets que permite tener bibliotecas en MIDP compartidas.

MIDP 3.0 también extiende la funcionalidad en varias áreas:

- Mejora la extensión y la expresión del interfaz de usuario.
- Mejor soporte para dispositivos que tienen pantallas grandes.
- Permite a los MIDlets dibujar en la pantalla secundaria.
- Mejor rendimiento en juegos.
- Almacenamiento seguro de registros.
- Almacenamiento persistente remoto.
- Incorpora soporte para IP V6.
- Soporta múltiples conexiones por dispositivo.

Por último, hay que tener en cuenta que también los sistemas operativos de los dispositivos tienen restricciones con respecto a MIDP. Por ejemplo, deben ofrecer un entorno de ejecución protegido donde la máquina virtual pueda correr, o algún tipo de apoyo para el acceso a una red, como puede ser el caso de un API para programar sockets, sobre el cual el protocolo HTTP se pueda implementar. Es el sistema operativo el que ofrecerá acceso al teclado y al posible dispositivo puntero, entregando los correspondientes eventos que surjan. Además, será el encargado de abstraer al MIDP la pantalla, ya que será visto por él como una matriz de pixeles, y de ofrecer un interfaz para el acceso al almacenamiento persistente.

MIDlets

Un MIDlet es la unidad básica de ejecución en el perfil MIDP, es decir, son las aplicaciones del perfil MIDP. Estas aplicaciones pueden utilizar tanto las facilidades aportadas por MIDP como las APIs que MIDP hereda de CLDC.

Un MIDlet consiste en una clase Java, que como mínimo, deriva de la clase abstracta `MIDlet`, y que se ejecutan en un entorno de ejecución dentro de la máquina virtual, que provee un ciclo de vida bien definido controlado mediante métodos de la clase MIDlet que cada MIDlet debe implementar.

Un grupo de MIDlets que están relacionados, es decir, que se ejecutan bajo el ámbito de una misma máquina virtual, se agrupan en un MIDlet suite. Todos estos MIDlet se empaquetan, instalan, desinstalan y borran como una única entidad y comparten recursos en tiempo de ejecución. Los MIDlets en este caso compartirán instancias de todas las clases de Java cargadas en la máquina virtual.

Para que un MIDlet pueda ser transferido a un teléfono móvil, previamente tienen que ser empaquetados en un fichero "jar". Junto con este fichero jar se encuentran otros dos ficheros importantes que son el fichero descriptor con extensión ".jad" (*Java Aplication Descriptor*) y un fichero denominado manifiesto con extensión ".mf" y en el que se describe el contenido del fichero ".jar".

En el fichero ".jad" se indican algunas características del MIDlet mediante la inclusión de atributos como `MIDlet-Name` (nombre del MIDlet), `MIDlet-Version` (versión del MIDlet), `MIDlet-Vendor`, `MIDlet<n>, name, icon, class` (diferentes características), `MIDlet-Jar-URL` (nombre del MIDlet con extensión .jar), `MIDlet-Jar-Size` (Tamaño del MIDlet), `MicroEdition-Profile` (Perfil empleado), `MicroEdition-Configuration` (configuración empleada).

Paquetes Opcionales de MIDP

La plataforma Java ME puede ampliarse combinando varios paquetes opcionales [6] con CLDC y CDC junto con sus perfiles. Los paquetes opcionales ofrecen un conjunto de API's para utilizar tecnologías existentes o emergentes como podrían ser bluetooth, servicios web, servicios multimedia, acceso a sensores…

La lista de paquetes opcionales disponibles para el perfil MIDP está formada por los Information Module Profile (IMP JSR 195), Wireless Messaging API (WMA JSR 120), Mobile Media API (MMAPI JSR 135), Location API for Java ME (JSR 179), SIP API for Java ME (JSR 180), Security and Trust Services API for Java ME (JSR 177), Mobile 3D Graphics (JSR 184), Java ME Web Services APIs (WSA JSR 172) y Bluetooth API (JSR 82). A continuación se hará una breve descripción de cada uno de ellos.

El *Information Module Profile* es un paquete que se puede combinar con CLDC y MIDP, que proporciona un entorno de aplicación para dispositivos empotrados que no tiene grandes capacidades gráficas. Proporciona la funcionalidad de aplicación básica para aplicaciones máquina-máquina, incluyendo conectividad de red, almacenamiento local y gestión del ciclo de vida de la aplicación.

El *Wireless Messaging API* es un paquete que se puede utilizar sobre CLDC y MIDP (1.0 y 2.0) y CDC y sus perfiles. Este API proporciona acceso independiente de la plataforma a recursos de comunicación sin cable como la mensajería SMS (*Short Message Service*). Existen dos especificaciones de este paquete opcional que son WMA 1.0, la especificación original a partir del JSR 120, Y WMA 1.1, especificación que incluye cambios para considerar el framework de seguridad y la arquitectura de comunicación de MIDP 2.0.

El *Mobile Media API* es un paquete que se puede utilizar sobre CLDC y MIDP (1.0 y 2.0) y CDC y sus perfiles. Este paquete extiende la funcionalidad de la plataforma Java ME incorporando soporte de audio y vídeo entre otros.

El *Location API for Java ME* es un paquete que se puede utilizar sobre CLDC 1.1 y CDC. Esta especificación permite la localización de dispositivos móviles para dispositivos con recursos limitados. El API se ha diseñado para generar información sobre la localización geográfica actual del terminal.

El *SIP API for Java ME* es un paquete que se puede utilizar sobre CLDC. El protocolo *Session Initiation Protocol* (SIP) se utiliza para establecer y gestionar sesiones IP multimedia. El mismo mecanismo se puede utilizar para proporcionar mensajería instantánea, presencia y servicios de juego. El API se ha diseñado para permitir que las aplicaciones Java envíen y reciban mensajes SIP.

El *Security and Trust Services API for Java ME* es un paquete que se puede utilizar sobre CLDC. Este paquete amplía las características de seguridad para la plataforma Java ME añadiendo APIs de cifrado, servicio de firma digital y gestión de credenciales de usuario.

El *Mobile 3D Graphics* es un paquete que se puede utilizar sobre CLDC y MIDP (1.0 y 2.0) y permite generar gráficos tridimensionales a frecuencias de imagen interactivas en dispositivos móviles. También incluye utilidades para la gestión de escenas 3D y animaciones.

El *Java ME Web Services APIs* es un paquete que se puede utilizar sobre CLDC y MIDP (1.0 y 2.0). Amplía la plataforma de servicios web para incluir Java ME. Estas APIs permiten que los dispositivos Java ME puedan ser clientes de servicios web mediante un modelo de programación consistente con la plataforma estándar de servicios web.

El *Bluetooth API* (JSR 82) es un paquete que se puede utilizar sobre CLDC y MIDP (1.0 y 2.0). Proporciona un estándar para la creación de aplicaciones Bluetooth, de forma que las aplicaciones desarrolladas con el paquete opcional puedan ejecutarse utilizando esta tecnología.

MSA

Para que los teléfonos móviles tengan una funcionalidad común y para que la portabilidad de las aplicaciones Java ME sea posible entre los dispositivos, en Java existe la *Mobile Service Architecture* (MSA) [7], que consiste en un conjunto de JSR's, las cuales deben ser soportadas por todos aquellos dispositivos que quieran ejecutar aplicaciones Java ME.

En las JSR 248 y JSR 249 se define una estructura global de API's destinadas a facilitar el desarrollo y despliegue de la mayor variedad de aplicaciones posible de la forma más portable y que alcance al mayor número de dispositivos.

La JSR 249 está basada en una actualización de la configuración CDC, mientras que la JSR 248 se basa en la configuración CLDC. Esta última MSA, la que está basada en la configuración CLDC, será más ágil y menos completa que la JSR 249, pero aún así, especifica un gran subconjunto de las capacidades que contiene la JSR 249.

Al igual que su predecesora, la JSR 185, MSA es una completa arquitectura para teléfonos móviles. Para garantizar una mayor compatibilidad entre las implementaciones, MSA también especifica una serie de aclaraciones que eliminan o reducen las ambigüedades que se han ido descubriendo en algunos JSR's. Como es un conjunto de JSR's, los dispositivos móviles que implementen la *Mobile Service Architecture*, podrán ejecutar las aplicaciones Java ME que se hayan basado en estas JSR's para su desarrollo. Por tanto, gracias a la MSA, los desarrolladores de aplicaciones Java ME pueden crear software portable para distintos dispositivos móviles.

La *Mobile Service Architecture* define dos pilas: una pila MSA completa que comprende dieciséis JSR's, y un subconjunto de ocho JSR's. Algunas de estas JSR's son obligatorias y otras son opcionales. Para cumplir con la MSA, una implementación debe soportar una JSR que sea obligatoria, o si es opcional y es relevante que la soporte. El ejemplo típico en este caso es el JSR 82 bluetooth API, el cual no es requerido siempre, pero debe estar soportado si el dispositivo en cuestión necesita contar con la tecnología bluetooth. Es decir, existen JSR's que son opcionales debido a que dependen de que el dispositivo cuente con el nivel hardware correspondiente; en el caso en que el dispositivo disponga del nivel hardware, la JSR's será obligatoria, como sucede en el caso de bluetooth.

Para que los dispositivos implementen el subconjunto MSA, deberán soportar las siguientes JSR's obligatorias:

- JSR 139 Java ME Connected Limited Device Configuration 1.1.
- JSR 118: Mobile Information Device Profile 2.1.
- JSR 135 Mobile Media API.
- JSR 184 Mobile 3D Graphics API.
- JSR 205 Wireless Messaging API.
- JSR 226 Scalable 2D Vector Graphics API.
- JSR 82 Java APIS for Bluetooth, en el caso en el que el dispositivo disponga de tecnología bluetooth.

Para implementar la pila MSA completa, los dispositivos deben soportar ocho JSR's más a parte de las ocho anteriores:

- JSR 172 (obligatoria) Java ME Web Services API.
- JSR 177 (obligatoria, opcional) Security and Trust Services API.
- JSR 179 (condicionalmente obligatoria) Location API.
- JSR 180 (obligatoria) SIP API.
- JSR 211 (obligatoria) Content Handler API.
- JSR 229 (obligatoria) Payment API.
- JSR 234 (obligatoria) Advanced Multimedia Supplements API.
- JSR 238 (obligatoria) Mobile Internationalization API.

2.1.1. Interfaces Gráficas de Usuario en Java ME

El perfil MIDP ofrece dos tipos de interfaz de usuario que son el “Interfaz de Bajo Nivel” y el “Interfaz de Alto Nivel”. La API de bajo nivel permite un control del interfaz de usuario más amplio que el API de alto nivel puesto que implementa varios eventos de la pulsación de las teclas y de la pantalla táctil, aunque no es un interfaz accesible y las aplicaciones que se basan en ella son menos portables que las basadas en la API de alto nivel debido a que tienen que tener en cuenta las características de la pantalla del dispositivo para el que son programadas. El interfaz de alto nivel por su parte, está orientado a widget (elementos que aparecen en la pantalla) como pueden ser cuadros de texto, listas seleccionables… lo que hace que sus aplicaciones sean accesibles y totalmente portables puesto que su implementación es independiente de la pantalla donde se vayan a visualizar. A pesar de estas dos ventajas, el interfaz de alto nivel tiene una importante desventaja que es el manejo de eventos, más reducido que en el api de bajo nivel.

Mientras que el interfaz de bajo nivel se basa en la clase abstracta `Canvas`, el interfaz de alto nivel está basado en la clase abstracta `Screen`, y ambos se encuentran dentro del paquete `javax.microedition.lcdui`. Este paquete está formado por dos clases básicas que son `Display` y `Displayable`. La clase `Display` se encarga de seleccionar el objeto `Displayable` que se muestra al usuario. La clase `Displayable` se encarga de la información a ser visualizada.

Clase Display

La clase `Display` cuenta con métodos para controlar la visualización de objetos `Displayable` y obtener propiedades del dispositivo como son el color, vibración… Solamente puede haber un `Display` por MIDlet.

Clase Displayable

La clase `Displayable` cuenta con dos subclases, `Screen` y `Canvas` a partir de las cuales se definen tres tipos de objetos: `Screen` con estructura predefinida, `Screen` genérico y `Canvas`.

Al primer tipo, `Screen` con estructura predefinida, pertenecen las subclases `Alert, List` o `TextBox` que encapsulan componentes de interfaces complejos que las aplicaciones no pueden enriquecer con nuevos componentes. En el tipo `Screen` Genérico se encuentra la clase `Form`, objeto que permite llenar la pantalla con texto, imágenes u objetos item. El tipo Canvas pertenece a la API de bajo nivel y en él el usuario tiene control total sobre los componentes del display y puede acceder a eventos de bajo nivel.

Los elementos que se encuentran dentro de la clase `Displayable` son:

- Clase `Command`: representa las opciones que se pueden ver en los botones del dispositivo móvil. Cuenta con atributos que identifican la naturaleza del objeto de tipo Command y que influyen en su posición en la pantalla del teléfono.
- Clase `Screen`: un `Screen` es el elemento funcional de la interfaz de usuario y cuenta con cuatro subclases que son: `Alert`, `List`, `TextBox` y `Form`.
  - Clase `Alert`: permite visualizar datos, ya sean texto o imágenes, durante un periodo de tiempo, es decir, según un timeout, antes de pasar a otra pantalla.
  - Clase `List`: la clase `List` implementa el interfaz `Choice` que define los tipos de listas existentes.

    Se emplea para que en la pantalla del terminal aparezca una lista de opciones, que pueden tener asociado un evento o no; es decir, puede generarse un evento cuando el usuario pulsa encima de una de las opciones (por ejemplo, puede aparecer un texto en la pantalla) o bien, emplearse simplemente para enumerar o visualizar una lista informativa.
  - Clase `TextBox`: esta clase permite al usuario introducir y editar texto, haciendo que éste se vea en toda la pantalla.

- Clase `Form`: la clase `Form` contiene un número arbitrario de elementos llamados ítems. Un `Item` solamente puede colocarse en un Form.
  - Clase `Item`: la clase `Item` dispone de un conjunto de elementos con una etiqueta asociada que pueden añadirse a un objeto `Form`.

  La clase `Form`, que es una superclase de la clase `StringItem` cuenta con los siguientes tipos de ítems: `ImageItem` que puede ser alineado, `TextField` para editar texto, `DataField` para visualizar fechas y horas.

  La clase `Item` además es una superclase de `ChoiceGroup`, item que implementa la interfaz `Choice` (no implícita). Es parecido a `List`, pero los eventos en este caso al pulsar una de las opciones no se gestionan mediante el método `ActionCommand` sino con el método `ItemStateChanged`. Esto tiene una implicación en la usabilidad que es que para acceder a la función configurada al producirse un evento en un displayable `List`, basta con pulsar una vez sobre la opción a seleccionar. Sin embargo, al emplear un item `ChoiceGroup`, hay que pulsar dos veces para tener el mismo efecto: una para seleccionar la opción y otra para confirmar.

- Interfaz `Choice`: el interfaz `Choice` define los tipos de listas que existen en el perfil de alto nivel. Estos tipos de listas son los siguientes:
  - `EXCLUSIVE` que permite tener seleccionado un único elemento de la lista simultáneamente.
  - `IMPLICIT` cuyo uso está permitido solamente en listas (`List`) donde el elemento que está enfocado es el que se selecciona implícitamente.
  - `MULTIPLE` que permite seleccionar cualquier número de elementos, incluido ninguno, y en cualquier combinación.
  - `POPUP` que no se puede usar con objetos `List` y tiene seleccionado exactamente un elemento a la vez.

2.1.2. Almacenamiento Persistente, RMS

MIDP define una sencilla base de datos orientada a registros que permite almacenar a las aplicaciones datos de forma persistente. Esta base se denomina *Record Management System* (RMS) [8].

El mecanismo básico de almacenamiento de RMS se denomina *record store*. Un *record store* es un conjunto de registros, y un registro es un byte array de datos de tamaño variable. Un *record store* está representado por un objeto de la clase `RecordStore`.

Existen reglas importantes sobre los record store:

1. El nombre de un *record store* consiste en una combinación de hasta 32 caracteres (sensible a las mayúsculas).

2. Los *record stores* creados por MIDlets de un mismo MIDlet suite están almacenados en el mismo espacio de nombres, y por lo tanto, pueden compartir y ver sus contenidos.
3. Los *record stores* creados por MIDlets en un MIDlet suite, no son accesibles para los MIDlets de otros MIDlets suite.
4. El nombre de un record store debe ser único en un MIDlet suite.

2.1.3. Conectividad HTTP en Java ME

En Java ME el soporte a conexiones de red viene dado por el Generic Connection Framework (GCF) [9] que define una serie de interfaces para dar soporte a los diferentes tipos de conexiones que pueden existir en los dispositivos móviles, pero no implementa ninguna de ellas.

El lugar donde se deben implementar estas interfaces es en los perfiles, y en el caso de los teléfonos móviles en el perfil MIDP. En MIDP se debe dar soporte obligatoriamente a conexiones HTTP implementando el interfaz `HttpConnection`. Este interfaz define los métodos y constantes necesarios para una conexión HTTP.

El GCF está formado por una sencilla jerarquía de interfaces y de clases para crear y gestionar conexiones HTTP, datagramas o flujos de datos. Como su nombre indica, el GCF proporciona un enfoque "genérico" para la conectividad, es decir, esta jerarquía de interfaces y de clases, se dice que es genérica puesto que proporciona un conjunto de API's comunes para todos los tipos básicos de conexiones, tanto para las conexiones basadas en la transmisión de paquetes como para las conexiones basadas en transmisión de flujos de datos. Esta generalización se hace posible gracias al uso de una jerarquía De interfaces que es extensible, una factoría de conexión y al uso de URL's que indican el tipo de conexiones que se van a crear. Gracias a que el GCF es extensible, los nuevos tipos de conexiones, definidos y estandarizzados por el *Java Community Process*, pueden ser añadidos como un nuevo subtipo de conexión soportado, proporcionando una nueva factoría de conexión (clase `Connector`) que soporta el nuevo tipo de conexión definido. También se define un nuevo formato de URL que identifica el nuevo tipo de conexión. Este nuevo tipo puede extender del interfaz básico de conexión o de uno de sus subtipos. Las URL's juegan un papel muy importante en el GCF, ya que describen la localización y el modo de acceso a un recurso en Internet usando una notación jerárquica. Por ejemplo, si se tiene la URL:

```
protocolo://usuario:contraseña@equipo/puerto/direccion_recurso
```

La información que se puede obtener es la siguiente:

- `protocolo` indica el protocolo o método de acceso como HTTP. En el GCF describe el tipo de conexión que se va a usar.
- `usuario` y `contraseña` son dos parámetros opcionales.
- `equipo` puede estar indicado por el nombre completo de dicho equipo o por su dirección IP.
- `puerto` es el puerto que se usará para la conexión y su interpretación dependerá del protocolo o método de conexión que se use.
- `direccion_recurso` es la dirección (*path*) donde se encuentra el recurso al que se quiere acceder.

El método de conexión relativo a este proyecto es HTTP [10], que es un protocolo de pregunta-respuesta en el cual, los parámetros de la pregunta deben establecerse antes de que se envíe dicha pregunta.

La conexión se fundamenta en tres estados, que son el estado *setup*, en el que los parámetros de la petición o pregunta pueden ser configurados, el estado *connected*, en el que se envían los parámetros de la petición y se espera la respuesta, y *closed*, el estado final, en el que se termina la conexión HTTP.

2.3 Bluetooth

Bluetooth [11] es una especificación para redes inalámbricas que permite la transmisión de voz y de datos entre dispositivos utilizando un enlace por radiofrecuencia.

La especificación para la comunicación inalámbrica bluetooth está desarrollada por el *Bluetooth Special Interest Group* (SIG) formado por las compañías 3Com, Ericsson, Intel, IBM, Agere, Microsoft, Motorola, Nokia y Toshiba.

2.3.1. Perfiles bluetooth

Un perfil [12] Bluetooth es la especificación de un interfaz de alto nivel para su uso entre dispositivos Bluetooth, de manera que para utilizar una cierta tecnología Bluetooth un dispositivo deberá soportar ciertos perfiles.

Los perfiles definen comportamientos generales que pueden utilizar los dispositivos para comunicarse. Por tanto, la forma de utilizar bluetooth está basada en los perfiles que soporta cada dispositivo.

La especificación de un perfil debe cubrir como mínimo los siguientes aspectos: dependencia con otros perfiles, formatos recomendados para la interfaz con el usuario, partes concretas de la pila Bluetooth que se utilizan como opciones particulares, parámetros, y también puede incluir una descripción del tipo de servicio requerido.

2.3.2. Protocolos bluetooth

La pila bluetooth está constituida por dos clases de protocolos [13]. Una primera clase llamada de protocolos específicos que implementa los protocolos propios de Bluetooth, y una segunda clase formada por el conjunto de protocolos adoptados de otras especificaciones. La pila de protocolos bluetooth se puede dividir en cuatro capas lógicas:

- Núcleo de Bluetooth que comprende Radio, Banda Base, LMP, L2CAP y SDP.

- Sustitución de cable que comprende RFCOMM.
- Protocolos adoptados que comprende PPP, UDP, TCP, IP, OBEX, WAP, IRMC, WAE.
- Control de telefonía que comprende TCS-binary, AT-Commands.

2.3.3 JSR 82- Bluetooth API

La especificación bluetooth define protocolos y perfiles de aplicación, pero no define ningún API. El JSR 82 [14] define API's que pueden ser usados para implementar determinados protocolos de bluetooth definidos en el volumen 1 de la especificación, y perfiles definidos en el volumen 2, y está destinada a proporcionar las siguientes capacidades:

- Registro de Servicios.
- Descubrimiento de dispositivos y de servicios.
- Establecimiento de conexiones RFCOMM, L2CAP y OBEX entre los dispositivos.
- Uso de las conexiones, envío y recepción de datos (la comunicación por voz no está soportada).
- Administración y Control de las comunicaciones de las conexiones.
- Proporciona seguridad para estas actividades.

El JSR 82 abstrae al programador de la complejidad de la pila de protocolos Bluetooth usando un conjunto de API's de Java que permiten al desarrollador centrarse en el desarrollo de la aplicación en lugar de en los detalles del nivel más bajo de la pila Bluetooth. Los API's de Java para Bluetooth no implementan la especificación Bluetooth sino que provee un conjunto de API's para acceder y controlar un dispositivo Bluetooth, y están basados en la versión 1.1 de la especificación Bluetooth consistiendo en dos paquetes opcionales: el core (núcleo) Bluetooth y *Object Exchange* (OBEX) API.

Los API's de Java para Bluetooth soportan dispositivos Bluetooth con las siguientes características:

- Un mínimo de 512 Kbytes de memoria total disponible (ROM-RAM) (los requisitos de memoria de la aplicación son adicionales).
- Conexión Bluetooth que implementen la configuración CLDC.

Arquitectura del API

El objetivo de la especificación fue proporcionar un API estándar, abierto y no propietario, que puede ser usado por todos los dispositivos Java ME. Por lo tanto fue designado usando el estándar de las API's Java ME y CLDC/MIDP.

Algunas características importantes son que la especificación proporciona soporte básico para los protocolos y perfiles Bluetooth, aunque no incluye API's específicos para todos los perfiles. El motivo es simplemente que el número de perfiles está en continuo crecimiento, y por tanto la especificación incorpora los protocolos de comunicación hacia cuyo uso están orientados todos los perfiles actuales de bluetooth.

Perfiles Bluetooth en Java ME

El sistema Bluetooth se basa en que los API's de Java deben cumplir con ciertos requisitos, y en concreto, debe cumplir con la implementación de los siguientes perfiles:

- *Generic Access Profile* (GAP),
- *Service Discovery Application Profile* (SDAP), y
- *Serial Port Profile* (SPP).

Además el sistema debe proporcionar un *Bluetooth Control Center* (BCC), un panel de control parecido a la aplicación que permite al usuario definir valores específicos para determinados parámetros de configuración en la pila; es decir, gestiona el acceso concurrente a las aplicaciones. El BCC Podría ser una aplicación nativa y una aplicación con una API separada, o simplemente un grupo de configuraciones que están especificadas por el fabricante y no pueden ser modificadas por el usuario. Hay que tener en cuenta que el BCC no es una clase o un interfaz definido en esta especificación pero es una parte importante de su arquitectura de seguridad.

Protocolos bluetooth implementados por Java ME

La plataforma Java ME implementa los protocolos bluetooth OBEX, L2CAP y RFCOMM y ofrece API's para su manejo.

OBEX

OBEX es la abreviatura de *Object Exchange* y facilita el intercambio de datos binarios entre dispositivos. Es similar a HTTP en diseño y funcionalidad, aunque difiere de él en algunos puntos como son el transporte, las transmisiones binarias y el soporte para realizar sesiones.

L2CAP

L2CAP son las siglas de *Logical Link Control and Adaptation Protocol*. L2CAP es utilizado dentro de la pila de protocolos de Bluetooth y se utiliza para pasar paquetes con y sin orientación a la conexión a sus capas superiores incluyendo tanto al *Host Controller Interface* (HCI) como directamente al gestor del enlace.

Las funciones de L2CAP incluyen segmentación y reensamblado de paquetes, Aceptando paquetes de hasta 64KB de sus capas superiores, multiplexación de varias fuentes de paquetes, proporción de una buena gestión para la transmisión unidireccional a otros dispositivos bluetooth, gestión de la calidad de servicio (QoS).

RFCOMM

RFCOMM es la abreviatura de *Radio Frequency Communication*. El protocolo RFCOMM es un conjunto simple de protocolos de transporte, construido sobre el protocolo L2CAP; y que proporciona sesenta conexiones simultáneas para dispositivos bluetooth emulando puertos serie RS-232.

A este protocolo a menudo se le llama emulación de puertos serie ya que el puerto serie de Bluetooth está basado en este protocolo.

BlueCove

BlueCove [15] es una implementación del JSR 82 para J2E actualmente compatible con Mac OS X, WIDCOMM, BlueSoleil y Microsoft Bluetooth stack de Windows XP SP2. Funciona bajo cualquier JVM desde la versión 1.1 para Windows Mobile, Windows XP, Windows Vista y Mac OS X.

Desde la versión 2.1 BlueCove se distribuye bajo la licencia de apache Software en su versión 2.0. Existe un soporte adicional en la versión 2.0.3 de BlueCove como licencia GPL para Linux llamada LinuxBlueZ.

BlueCove proporciona el api del JSR82 para J2E.

2.4. Glucómetro o Medidor de Glucosa

Un medidor de Glucosa o Glucómetro [16] es un dispositivo médico que se utiliza para la determinación de la glucosa en la sangre. Es un elemento básico para la medición de la glucosa para personas con diabetes o hipoglucemia (bajo nivel de glucosa en sangre).

Una pequeña gota de sangre, obtenida pinchando la piel con una lanceta, es colocada en una tirilla desechable, la cual el medidor leerá y utilizará para calcular el nivel de glucosa en la sangre. Estos medidores muestran el valor en mg/dl (mg de glucosa por dl de sangre) o mmol/l.

2.4.1. Características de un Glucómetro

Hay varias características fundamentales que variarán de modelo a modelo:

- **Tamaño**: El tamaño medio de estos dispositivos es aproximadamente el de la palma de la mano, de todas formas los hay más pequeños y más grandes. Pueden ser alimentados a batería o a corriente eléctrica.
- **Tiras de Muestreo**: Es un elemento consumible que contiene sustancias químicas que reaccionan con la glucosa presente en la gota de sangre, una tira diferente es utilizada para cada medición. Algunos modelos utilizan una tira plástica con una pequeña porción impregnada con glucosa oxidasa y

otros componentes. Cada tira es utilizada una vez y luego descartada. Algunos modelos en lugar de tiras utilizan discos de varias tiras que pueden ser usados para varias lecturas.

- **Calibración**: Debido a que las tiras pueden variar de lote a lote, algunos modelos requieren que se les introduzca un código disponible en el prospecto de la caja de tiras o en un chip que viene dentro de la caja de tiras. Al introducir el código o el chip en el medidor de glucosa, el medidor quedará calibrado para ese lote de tiras. Si el proceso de calibración fuese realizado de forma incorrecta, la desviación puede llegar a ser de 4mmol/L. Las implicaciones en el uso de un medidor calibrado incorrectamente pueden llegar a ser serias para pacientes con diabetes. Para medidores descalibrados, la probabilidad de aplicar un error de 2 unidades en la aplicación de una dosis de insulina es de un 50 % y la probabilidad de un error de 3 unidades es del 24 % comparado con un 0,49% cuando se utiliza un medidor correctamente calibrado.
- **Volumen de la muestra de sangre**: El tamaño de la gota de sangre necesaria varía, dependiendo del medidor entre 0.3 y 1 µl. (Modelos mas antiguos requieren de volúmenes mayores) El requisito de un menor volumen reduce la frecuencia de pinchazos improductivos.
- **Tiempo de Medición**: Es el tiempo que le toma al medidor calcular el valor de glucosa en sangre. Dependiendo del modelo podrá variar de 3 a 60 segundos.
- **Display**: El valor de glucosa en sangre en mg/dl o mmol/l es presentado en un pequeño display. Las unidades de medida preferidas varían en cada país, para pasar el valor de mmol/l a mg/dl hay que multiplicar por 18 y para pasar mg/dl a mmol/l hay que dividir entre 18. En la gran mayoría de los equipos puede seleccionarse la unidad de medida.

2.4.2. Tipos de glucómetros

Existen dos tipos de glucómetros según la metodología de medición:

- **Reflectómetro**: estos equipos miden la luz que es reflejada desde el reactivo después de experimentar una reacción química (oxidación enzimática de la glucosa). El resultado de la reacción es un compuesto cromático. El color resultante es proporcional a la cantidad de glucosa presente.
- **Biosensores**: Estos equipos corresponden a la nueva tecnología, miden la corriente eléctrica generada por la sangre presente en el reactivo (corriente eléctrica generada por la oxidación de la glucosa)

2.5. Síntesis de Voz

En esta sección se va a hacer una pequeña introducción a la síntesis de voz [17] y a los diferentes tipos de sintetizadores de voz existentes. También habrá una sección que intentará diferenciar entre sintetizador de voz y lector de pantalla, puesto que es

importante entender qué es un sintetizador y qué es un lector de pantalla para comprender el desarrollo del proyecto, pero no se aportará nada nuevo sobre estas tecnologías.

2.5.1. Introducción a la Síntesis de Voz

La voz sintética es una voz artificial (no pregrabada), generada mediante un proceso de sintetización del habla. La síntesis de habla es la producción artificial de habla humana. Un sistema usado con este propósito recibe el nombre de sintetizador de habla y su capacidad de convertir texto en habla puede llevarse a cabo en software o en hardware. La síntesis de voz se llama a menudo en inglés text-to-speech (TTS), en referencia a su capacidad de convertir texto en habla. Sin embargo, hay sistemas que en lugar de producir voz a partir de texto lo hacen a partir de representación lingüística simbólica. La calidad de una voz sintética vendrá dada por dos factores:

1. Su inteligibilidad: ¿con qué facilidad/dificultad es entendida?
2. Su naturalidad: ¿en qué medida se asemeja a la voz real de un humano?

Un sistema texto a voz se compone de dos partes: un front-end y un back-end. A grandes rasgos, el front-end toma como entrada texto y produce una representación lingüística fonética. El back-end toma como entrada la representación lingüística simbólica y produce una forma de onda sintetizada; es decir, toma la representación lingüística simbólica y la convierte en sonido. El back-end se llama a menudo sintetizador.

2.5.2. Tecnologías de Síntesis de Voz

Las dos características utilizadas para describir la calidad de un sintetizador de voz son la naturalidad e inteligibilidad. La naturalidad de un sintetizador de voz se refiere a hasta qué punto suena como la voz de una persona real. La inteligibilidad de un sintetizador se refiere a la facilidad de la salida de poder ser entendida. El sintetizador ideal debe de ser a la vez natural e inteligible, y cada tecnología intenta conseguir el máximo de ambas. Algunas de las tecnologías son mejores en naturalidad o en inteligibilidad y las metas de la síntesis determinan a menudo qué aproximación debe seguirse.

Hay dos tecnologías principales usadas para generar habla sintética: síntesis concatenativa y síntesis de formantes.

La síntesis concatenativa se basa en la concatenación de segmentos de voz grabados. Generalmente, la síntesis concatenativa produce los resultados más naturales. Sin embargo, la variación natural del habla y las técnicas automatizadas de segmentación de formas de onda resultan en defectos audibles, que conllevan una pérdida de naturalidad.

Hay tres tipos básicos de síntesis concatenativa:

- Síntesis por Selección de Unidades. La síntesis por selección de unidades utiliza una base de datos de voz grabada (más de una hora de habla grabada). Durante la creación de la base de datos, el habla se segmenta en las siguientes unidades: fonemas, sílabas, palabras, frases y oraciones.

  Normalmente, la división en segmentos se realiza usando un reconocedor de voz modificado para forzar su alineamiento con un texto conocido. Después se corrige manualmente, usando representaciones como la forma de onda y el espectrograma.
- Síntesis de Difonos. La síntesis de difonos usa una base de datos mínima conteniendo todos los difonos que pueden aparecer en un lenguaje dado. El número de difonos depende de la fonotáctica del lenguaje: el español tiene unos 800 difonos, el alemán unos 2500. En la síntesis de difonos, la base de datos contiene un sólo ejemplo de cada difono.

  La calidad del habla resultante es generalmente peor que la obtenida mediante selección de unidades pero más natural que la obtenida mediante sintetización de formantes. La síntesis de difonos sufre los defectos de la síntesis concatenativa y suena robótica como la síntesis de formantes, y tiene pocas ventajas respecto a estas técnicas aparte del pequeño tamaño de la base de datos.

- Síntesis Específica para un Dominio. La síntesis específica para un dominio concatena palabras y frases grabadas para crear salidas completas. Se usa en aplicaciones donde la variedad de textos que el sistema puede producir está limitada a un particular dominio, como anuncios de salidas de trenes o información meteorológica.

  La síntesis de formantes no usa muestras de habla humana en tiempo de ejecución. En lugar de eso, la salida se crea usando un modelo acústico. Parámetros como la frecuencia fundamental y los niveles de ruido se varían durante el tiempo para crear una forma de onda o habla artificial. Este método se conoce también como síntesis basada en reglas.

  Muchos sistemas basados en síntesis de formantes generan habla robótica y de apariencia artificial, y la salida nunca se podría confundir con la voz humana.

  La síntesis de formantes puede ser muy inteligible, incluso a altas velocidades, evitando los defectos acústicos que pueden aparecer con frecuencia en los sistemas concatenativos. La síntesis de voz de alta velocidad es a menudo usada por las personas con discapacidad visual para utilizar ordenadores con fluidez.

  Por otra parte, los sintetizadores de formantes son a menudo programas más pequeños que los sistemas concatenativos porque no necesitan una base de datos de muestras de voz grabada. De esta forma, pueden usarse en sistemas empotrados, donde la memoria y la capacidad de proceso son a menudo exiguas. Por último, dado que los sistemas basados en formantes tienen un control total sobre todos los aspectos del habla producida, pueden incorporar una amplia variedad de tipos de entonaciones, que no sólo comprendan preguntas y enunciaciones.

2.5.3. Diferencias entre Sintetizador de Voz y Lector de Pantalla.

Un lector de pantalla es una aplicación software que trata de identificar e interpretar aquello que se muestra en pantalla. Esta interpretación se representa a continuación al usuario mediante sintetizadores de texto a voz, iconos sonoros, o una salida braille.

En definitiva y por dar una explicación sencilla, el lector de pantalla es el software encargado de interpretar los objetos que aparecen en una pantalla de un ordenador como iconos, editores de texto, navegadores web… mientras que el sintetizador de voz es la aplicación que convierte aquello que ha sido capaz de interpretar el lector de pantalla a voz.

El tipo de síntesis de voz que se emplea para crear los sintetizadores usados por los lectores de pantalla es la síntesis de formantes, que como se ha visto, se basa en generar voz perfectamente inteligible a pesar de contar con un cierto tono robotizado.

2.6 Aplicaciones de Telemedicina

La Telemedicina [18] incluye tanto el diagnóstico como el tratamiento a distancia de las patologías. La telemedicina hace posible optimizar los recursos sanitarios y acercarlos a lugares donde es difícil que llegue alguno de ellos, como puede ser la atención de los pacientes por especialistas. A continuación se enumeran algunas aplicaciones de telemedicina que actualmente se están llevando a cabo:

- Sistema de Telemedicina de Roche Emminens Conecta [19]: sistema por el cual un paciente diabético emplea el medidor de glucosa de Roche Accu-Check Compact Plus, el cual se conecta vía infrarrojos con una aplicación instalada en un teléfono móvil para posteriormente enviarlo a un servidor web utilizando tecnología sms.
  Como se puede ver, esta aplicación de telemedicina comparte objetivos con el prototipo desarrollado en este proyecto, aunque el objetivo principal no es el de tener una aplicación móvil accesible, sino el de tener una aplicación de telemedicina que ayude a reducir los tiempos de espera en la consulta médica con soluciones propietarias.
- Teleconsulta Neumológica [20]: sistema de teleconsulta que incorpora dos pruebas respiratorias: la radiografía de tórax y la espirometría forzada. Esta aplicación se lleva a cabo en el Servicio de Neumología del Hospital San Pedro de Alcántara, Cáceres.
- Telemonitorización de insuficiencias cardiacas [21]: sistema por el cual una persona que sufre pequeños paros cardiacos recibe descargas automáticamente gracias a un dispositivo instalado en su cuerpo. Este dispositivo cuenta con un registro de sucesos que se puede descargar a un PC mediante comunicación inalámbrica con un aparato que se ubica sobre el paciente y recibe los datos. Esta aplicación se emplea en el Hospital Txagorritxu de Vitoria-Gasteiz.

2.7. Conclusiones Estado del Arte

Después de haber analizado y estudiado las diferentes tecnologías que se van a emplear en la realización del prototipo, puede concluirse que Java ME es un lenguaje adecuado para desarrollar aplicaciones móviles accesibles para un lector de pantalla puesto que tiene un interfaz gráfico de usuario, el Interfaz de Alto Nivel, cuyos elementos así lo permiten.

Por otra parte, emplear RMS como sistema de almacenamiento persistente de los controles y HTTP como protocolo para enviar los datos al servidor web, implica que la aplicación sea muy portable ya que son dos paquetes soportados por todos los teléfonos móviles.

También se puede concluir que el JSR 82- Bluetooth API es un API altamente soportado por todos los teléfonos móviles que cuenten con un dispositivo bluetooth incorporado, y que este puede ser un factor a tener en cuenta en caso de optar entre esta tecnología inalámbrica u otras como podría ser infrarrojos.

Capítulo 3. Descripción de la aplicación

En este capítulo se abordará en primer lugar el funcionamiento de la aplicación de forma general, para posteriormente introducirse en el funcionamiento de cada una de las tres aplicaciones que forman el prototipo.

3.1. Funcionamiento de la aplicación

El funcionamiento en conjunto ofrece la posibilidad a una persona ciega o deficiente visual la oportunidad de hacerse una medición de glucosa con un "glucómetro" y posteriormente almacenar dicha medición en su teléfono móvil transmitiendo los datos vía bluetooth desde un PC, haciendo uso de la aplicación servidora, para posteriormente y desde el teléfono móvil, enviarlos vía HTTP a un servidor web encargado de almacenarlos y mostrarlos. El prototipo se basa en tres aplicaciones: aplicación móvil, aplicación servidora y aplicación web.

La aplicación móvil está basada en el perfil MIDP 2.0 de la configuración CLDC 1.1 de Java ME, mientras que para desarrollar las aplicaciones servidora y web se ha empleado el JDK 1.6.

El empleo como soporte de una aplicación servidora ejecutada en un PC se debe a que ésta hace de puente entre los datos transferidos desde el medidor de glucosa al PC, y su posterior envío al teléfono móvil vía bluetooth. Cabe destacar que esta aplicación servidora, desarrollada para su ejecución en un PC, tiene como fin su implantación en un medidor de glucosa para que éste se pueda comunicar con el teléfono móvil sin necesidad de más dispositivos intermediarios.

Por tanto, el usuario se hará una medición de glucosa con su "glucómetro", transferirá los datos al PC con las herramientas propias del medidor, y finalmente las enviará por bluetooth a su teléfono móvil ejecutando la aplicación servidora. Una vez se han recibido los datos en el teléfono móvil, el usuario podrá introducir un comentario asociado a dicha medición antes de almacenarlo. La aplicación móvil es una agenda de controles los cuales pueden ser consultados por día, mes, o todos los que estén almacenados. En el caso de que el usuario lo decida, existe una opción para enviar los datos vía HTTP a un servidor al que accederá su endocrino y podrá ver los controles de su paciente.

La accesibilidad es un asunto transversal en el desarrollo de todo el prototipo de manera que se tiene una aplicación íntegramente accesible para personas ciegas o con discapacidad visual. Las personas ciegas o con los mismos requisitos que éstas, emplean un lector de pantalla para interactuar con las diferentes funcionalidades que un teléfono móvil o un ordenador tiene. En este caso, el lector de pantalla empleado en el teléfono móvil es el *Mobile Speak de Codefactory* y en el PC el *JAWS de Freedom Scientific*. Gracias a la implementación de la aplicación móvil, el usuario ciego o deficiente visual puede acceder a todas las características y opciones de la misma.

Con respecto a la aplicación web, la accesibilidad en este campo es algo que está más desarrollado aunque no por ello se ha dejado de lado.

3.2. Aplicación Móvil

En esta sección se desarrollará el funcionamiento de la aplicación móvil y de las opciones que la conforman, así como su estructura de clases y de métodos.

3.2.1. Descripción de la Aplicación Móvil

La aplicación cuenta con un interfaz de usuario sencillo en cuya vista inicial ofrece cinco opciones:

- Ver instrucciones.
- Vista mensual.
- Vista diaria.
- Obtener datos vía bluetooth.
- Enviar datos vía HTTP.

Como se viene comentando, la accesibilidad de la aplicación ha sido un factor determinante en la elección del tipo de interfaz. En efecto, el interfaz de bajo nivel de Java ME se limita a dibujar a nivel de píxel la información que aparece en la pantalla. Esto para un lector de pantalla, por decirlo de una manera sencilla, es como si se creasen imágenes con extensión .jpg lo que es imposible leer por parte de un programa de estas características. Por tanto, el interfaz empleado es el de alto nivel que se limita a utilizar widgets, y que a pesar de no poder explotar todas las características del terminal móvil, es completamente accesible si los elementos que lo conforman se usan de una manera adecuada.

Con "adecuada" se quiere decir que si bien el uso de elementos pertenecientes a la clase `Displayable` es accesible por parte de un lector de pantalla, hay que hacer uso del sentido común para utilizarlos correctamente de acuerdo con la información que se desee que transmitan. Por ejemplo, un lector de pantalla puede leer el título de un objeto `Displayable` (título de un `Form`, de un `TextBox`, de un `Alert`) o de un `List` pero solamente lo leerá cuando el terminal muestre la vista, y por tanto, el lector de pantalla no podrá volver a interactuar con el título correspondiente. Si se tiene esto en cuenta, se deberá utilizar el título de un elemento `Displayable` exclusivamente para indicar el nombre de la vista y no para aportar información imprescindible para el manejo de la aplicación.

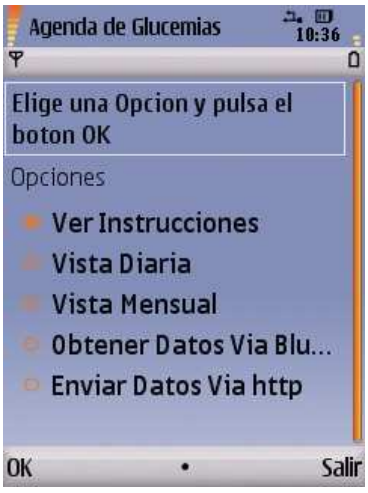
Como se acaba de comentar, el desarrollador ha de hacer uso del sentido común para decidir qué información es la relevante e imprescindible para posteriormente, actuar en consecuencia y emplear el objeto `Displayable` adecuado en unas ocasiones, o un `Item` en otras. En efecto, si el desarrollador considera crítico el acceso a una determinada información como podría ser la hora o fecha actual para su utilización en un paso posterior de la aplicación, lo correcto sería utilizar un `Item` `TextField` o `DateField` de un `Displayable Form` para que el usuario pueda interactuar y acceder a la información tantas veces como necesite con las flechas del cursor de su terminal móvil.

En las siguientes secciones se verá que para crear una vista en la que aparezca una lista con diferentes opciones para seleccionar, el `Displayable` empleado es un `List` y también se podrá leer el motivo por el que se decide. En la vista principal sin embargo, se ha optado por un `ChoiceGroup` dentro de un `Displayable Form` para visualizar las cinco opciones con que cuenta la aplicación. El motivo por el que se usa un `ChoiceGroup` es que es un objeto `Item` de un `Displayable Form`, lo que implica que puede ir acompañado de otros ítems, como en este caso ocurre con un item `TextField`.

Por tanto, en este prototipo se ha decidido incluir un `TextField` con un mensaje cuyo contenido invita al usuario a seleccionar una de las cinco opciones ya que se ha optado por pensar que al usuario le resultaría más intuitivo el tener un mensaje inicial que le indique cómo usar la aplicación, que el que aparezca directamente una lista con las opciones. Si se hubiese optado por un `Displayable List` el usuario solamente podría interactuar con las opciones y no con el título ni con otro `Item` puesto que este elemento no permite incluir ítems. Del mismo modo, si se hubiese utilizado un `ChoiceGroup` como único `Item` del `Displayable Form`, el lector de pantalla verbalizaría el título del `ChoiceGroup` solamente la primera vez que se lanza la ventana, de manera que el usuario no puede volver a interactuar con el mismo. Sin embargo, al emplear un `Displayable Form` con dos ítems, un `TextField` y un `ChoiceGroup`, el usuario puede interactuar tanto con el texto del `TextField` como con sus opciones (opciones del `ChoiceGroup`), las cuales deberán ser seleccionadas primero y confirmadas a continuación pulsando el mismo botón (botón de selección central en el caso del teléfono móvil utilizado para las pruebas).

Una vista del resultado del uso de un item `TextField` junto con un item `ChoiceGroup` se puede ver en la figura 1.

Figura 1.Vista Principal.

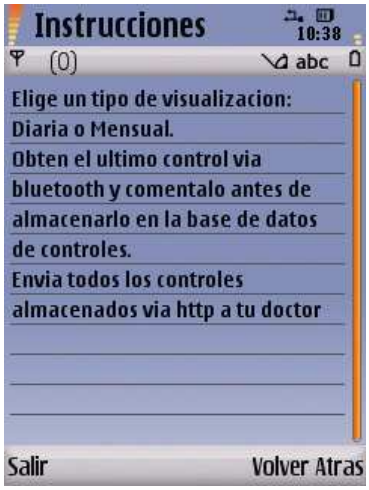


Ver Instrucciones

Esta opción está creada empleando un elemento `TextBox` de Java ME y es completamente accesible. Al introducir el carácter de retorno de carro en la descripción del funcionamiento de la aplicación, se consigue que el usuario pueda navegar por las distintas líneas que aparecen en la pantalla de su teléfono móvil empleando las teclas del cursor de su teclado estándar.

En la figura 2 se puede ver la vista de las instrucciones.

Figura 2. Vista de las Instrucciones.



Vista Diaria

Como se ve en la figura 3, si el usuario selecciona esta opción accederá a una vista en la que hay un `Displayable List` con dos opciones: "Todos los controles almacenados" y "Elegir Día del Mes Actual".

Figura 3. Vista Diaria.

Todos los controles almacenados

Si el usuario elige la primera opción, “Todos los controles Almacenados”, elección que se puede apreciar en la figura 4, podrá ver un listado de todas las mediciones de glucosa que se han ido almacenando con la siguiente información:

- Mes: tres primeras letras del mes en inglés en el que se ha tomado la medición.
- Día: día en formato `dd`.
- Tiempo de la medición: hora y minutos en formato `hh:mm`.
- Año: año en formato `aaaa`.
- Nivel de glucosa: valor decimal.
- Comentario: cadena de caracteres asociada a la medición.

La separación elegida entre los campos es un espacio ya que ha resultado la más cómoda de escuchar cuando es pronunciada por el lector de pantalla.

Otro factor destacable es la elección de un `Displayable List` para la visualización de los controles y no de un `TextBox` en el que apareciese cada control en una línea ya que debido a las características del lector de pantalla, cuando éste accede a un `Displayable List`, la información que reproduce es la siguiente:

```
1.- “Jan 23 9:48 2010 150 Normal”.
2.- “1 de 22”.
```

En el punto de información 1, el usuario puede escuchar todos los campos almacenados y asociados a la medición de glucosa.

En el punto de información 2, el lector de pantalla avisa que es el primer elemento de un `Displayable List` de un total de 22. Gracias a esto, el usuario puede saber en qué control está y cuántos tiene almacenados en total.

De aquí en adelante, cuando se mencione que el elemento elegido para mostrar la información es un `Displayable List`, el motivo de la elección del mismo será siempre el que se acaba de comentar.

Figura 4. Vista de Todos los controles.

Elegir Día del Mes Actual

Si el usuario lo prefiere, puede consultar los controles almacenados en un día concreto del mes en curso. Por ejemplo, si se encontrase en el mes de Mayo, cuando el usuario seleccione esta opción accederá a un `Displayable List` con 31 días; podrá navegar de forma totalmente accesible entre los días del mes, y seleccionar aquel del que quiera conocer los datos almacenados.

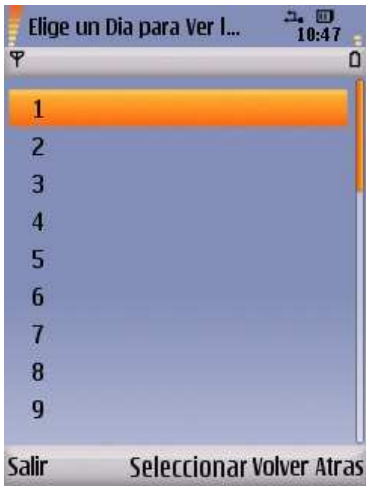
Una vez se ha seleccionado el día, aparecerá otro `Displayable List` con los controles correspondientes. En el caso de que no haya nada almacenado, aparecerá un mensaje con el siguiente texto:

```
(No hay datos)
```

Este texto es gestionado por el propio teléfono móvil y puede ser leído por el lector de pantalla. La vista de la aplicación permanecerá en esta situación hasta que el usuario pulse el botón de “Salir” o “Volver a Atrás”.

En la figura 5 se puede ver una vista de los días del mes en curso.

Figura 5. Vista de los Días del Mes Actual.



Vista Mensual

Cuando se selecciona esta opción se accede a una lista en la que aparecen los meses del año, como se puede apreciar en la figura 6. El elemento del interfaz de alto nivel que se ha empleado para visualizar los meses es un `Displayable List` y es completamente accesible. El usuario puede navegar por la lista empleando las teclas del cursor y seleccionar un mes pulsando en el botón de confirmación de su teclado.

Cuando el usuario pulsa un mes, aparece otra lista con los días que componen ese mes. Finalmente, si el usuario entra dentro de un día, puede ver los controles almacenados en ese día.

Figura 6. Vista Mensual.

Obtener Datos Vía bluetooth

En esta sección primero se hablará de forma rápida acerca de las diferentes vistas que van apareciendo al usuario, para posteriormente incidir en los pasos en los que se basa una aplicación bluetooth y los eventos generados por la misma que son quienes provocan la visualización de las distintas vistas.

Cuando el usuario selecciona esta opción pueden suceder dos cosas en el caso de que el dispositivo bluetooth esté activado previamente o no. Como se ve en la figura 7, si no está activado, aparece una alerta gestionada por el propio teléfono indicando al usuario que la aplicación requiere de la activación del bluetooth, y pregunta si se desea activar. En el caso en el que el bluetooth esté activado previamente, el mensaje anterior no aparecerá.

Figura 7. Mensaje de Activación Bluetooth.

En cualquier caso, se acaba accediendo a una vista en la que aparece un `TextField` que indica que se están buscando dispositivos bluetooth tal y como aparece en la figura 8. Este texto puede ser leído sin problemas por el lector de pantalla aunque no el título de la ventana que se abre; es decir, el título asociado a la ventana que se abre es leído por el lector de pantalla en el momento de su lanzamiento, pero luego el usuario no puede interactuar con él. Sin embargo, con el texto que aparece en el `TextField` si se puede interactuar y volver a leer cuando se pulsan las teclas del cursor del dispositivo móvil.

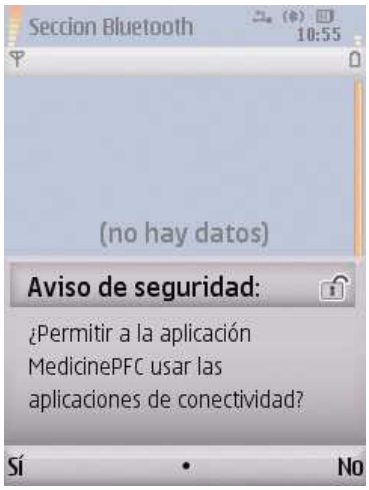
Figura 8. Vista de la Búsqueda de Dispositivos.

Cuando el teléfono ha detectado los dispositivos bluetooth, aparece una vista con un Displayable List que lista todos los dispositivos detectados (figura 9).

Figura 9. Vista de los Dispositivos Detectados.

Cuando se selecciona el dispositivo en el que está la aplicación servidora corriendo, el teléfono vuelve a sacar un mensaje por pantalla en el que se le consulta al usuario que se necesita activar la transmisión de datos. A continuación se ve la vista del mensaje gestionado por el teléfono móvil en la figura 10.

Figura 10. Vista de la Autorización de Conectividad.



Tras la confirmación, la recepción de datos comienza y el usuario del teléfono móvil sabe cuándo se han recibido todos los datos ya que una vez ha sucedido esto, aparece una vista en la que se visualizan los datos recibidos junto con un `TextField` vacío y editable para introducir el comentario asociado a la medición de glucosa recién recibida, tal y como se aprecia en la figura 11.

Figura 11.Vista de los Datos Recibidos.

Si se observa la figura 12, se puede comprobar que una vez se ha introducido el comentario, el usuario debe pulsar el botón "Introducir Comentario" en cuyo caso, saltará una alerta creada con un `Displayable Alert` durante un tiempo suficiente como para que el usuario pueda leer el texto incluido dentro de la misma. Esta alerta indicará al usuario que los datos se han guardado correctamente.

Figura 12. Vista de la Alerta comentario Introducido.

A continuación se explicarán los métodos de callback de una aplicación bluetooth. En esta explicación se utilizarán las características y requisitos del JSR 82 como base para entrelazarlo con las características de accesibilidad de la aplicación. Es decir, se abordará la necesidad de implementación de cuatro métodos de "`callback`" y el tratamiento de sus resultados utilizando el interfaz de Alto Nivel.

El desarrollo de una aplicación bluetooth en Java ME se puede resumir en tres pasos:

1. Comienzo del proceso de descubrimiento de dispositivos.
2. Consultas a los dispositivos descubiertos en el proceso 1.
3. Inicio y proceso del intercambio de datos.

1. Comienzo del proceso de descubrimiento de dispositivos:

Este proceso se usa para decir a la pila bluetooth del dispositivo que comience a buscar dispositivos bluetooth próximos a él y que lógicamente estén disponibles. En el caso de un MIDlet, la implementación de la pila bluetooth la provee el proveedor del servicio del dispositivo móvil al JSR 82. Este proceso de descubrimiento es iniciado por el `Discovery Agent` (agente de descubrimiento de dispositivos) presente en el propio dispositivo móvil.

```
// Obtención del "Discovery Agent"
DiscoveryAgent agent =
LocalDevice.getLocalDevice().getDiscoveryAgent();

// comienzo de las consultas al "Discovery Agent"
agent.startInquiry(inquiryMode, this);
```

Una vez el "Discovery Agent" ha comenzado el proceso de descubrimiento de dispositivos, llamará a varios métodos de `callback` que deben ser incluidos en la clase que debe implementar el interfaz `DiscoveryListener`.

En concreto se deben implementar cuatro métodos de este interfaz. Dos de ellos se basan en el proceso de descubrimiento de dispositivos:

```
deviceDiscovered(RemoteDevice btDevice, DeviceClass cod)
inquiryCompleted(int discType)
```

Estos dos métodos se emplearán para añadir al interfaz de usuario los dispositivos descubiertos así como para indicar que el proceso ha finalizado. Aquí es donde entra el interfaz de Alto Nivel y la utilización de uno de sus Displayables para mostrar la lista de dispositivos descubiertos. En concreto, en el método `inquiryCompleted`, se actualiza un `Displayable List` con los dispositivos descubiertos. Esto permitirá al usuario ciego o deficiente visual navegar con las teclas del cursor entre los diferentes dispositivos y seleccionar pulsando el botón "entrar" (normalmente el botón central del Joystick).

2.- Descubrimiento de los servicios en los dispositivos descubiertos

Puesto que lo que se quiere es transmitir datos, se necesita descubrir los servicios en los dispositivos que permitan conseguir este objetivo. Para ello, se necesitan especificar los atributos correctos y los UUIDS (identificadores de usuario) en el proceso de descubrimiento de servicios.

```
// Identificador bluetooth del dispositivo.
UUID uuid = new UUID(0x1101);
```

En este proceso, se utilizarán los dos métodos restantes del interfaz `DiscoveryListener:`

```
servicesDiscovered(int transId, ServiceRecord[] records)
serviceSearchCompleted(int transID, int respCode)
```

El primero de los dos métodos inicia un hilo que se encarga de comenzar el intercambio de datos con el servidor bluetooth. Por tanto, parece razonable pensar que en este proceso se necesitará la URL para conectarse con un servicio en concreto que pertenece a su respectivo dispositivo. Esta URL será usada para establecer la conexión bluetooth para transmitir los datos y es una dirección hardware del dispositivo bluetooth.

3.- Envío de Datos:

En este prototipo el envío de datos comienza en el momento en que el usuario selecciona el dispositivo correspondiente en el `Displayable List`. Debido a que la cantidad de datos que se transmite desde la aplicación servidora al terminal móvil es pequeña, no ha sido necesario implementar una vista específica con un mensaje indicando el proceso de transmisión de datos. Por tanto, el usuario tiene la confirmación de que ha recibido los datos en el terminal puesto que en el momento que se han recibido todos ellos, aparece un `Displayable Form` que contiene un `TextField` por cada campo recibido y un último `TextField`, esta vez editable, para que el usuario introduzca el comentario asociado al control de glucosa.

Si el usuario no introduce ningún comentario, caso permitido por la aplicación, en el `RecordStore` se almacenará el control cuyo contenido del campo comentario será "No hay Comentario".

Envío de datos vía HTTP

Cuando el usuario elige esta opción, todos los controles de glucosa que estén almacenados en el teléfono móvil se enviarán vía HTTP a un servidor cuya URL estará prefijada en la aplicación móvil.

3.2.2. Estructura de Clases y de Métodos

En esta sección se verá una relación de las clases que forman la aplicación móvil así como los métodos de estas clases; en concreto, se podrán ver dos tipos de tablas; en la tabla de clases aparecerá el nombre de la clase, las clases de las que hereda, los interfaces que implementa y una pequeña descripción. En la tabla de métodos, una por cada clase implementada, aparecerá el nombre del método, una indicación de si es un método propio o relativo a algún paquete y una pequeña descripción del objetivo de la misma.

Relación de Clases

La aplicación móvil está formada por tres clases: la clase principal que crea el MIDlet llamada `MainMIDlet`, la clase encargada de gestionar el almacenamiento persistente llamada `RegisterData`, y la clase que se encarga de gestionar la obtención de los datos almacenados en el terminal móvil y enviarlos por http, a la que se le ha llamado `SendData`.

Tabla 1. Tabla de clases de la aplicación móvil.

CLASE	HEREDA DE	INTERFACES QUE IMPLEMENTA	DESCRIPCIÓN
`MainMIDlet`	MIDlet	`CommandListener, DiscoveryListener`	Clase principal de la aplicación.
`RegisterData`	-	`RecordFilter`	Clase que se encarga del almacenamiento y recuperación de los datos.
`SendData`	-	-	Clase que se utiliza para el envío de los datos por HTTP.

Relación de Métodos

En este apartado se podrá ver una tabla por cada clase que forma la aplicación móvil precedidas por una explicación de los formatos empleados y que resulta más interesante comentar.

Métodos de la clase `MainMIDlet`

En esta clase es interesante destacar el formato en el que se reciben los datos por la conexión bluetooth. Para obtener los datos se ha empleado un flujo de datos de entrada (`DataInputStream`) el cual contiene todos los datos asociados a un control (mes, día, hora, minutos, año y nivel de glucosa) en formato `String`, para lo que ha tenido que usarse el método `readUTF()` de la clase `DataInputStream`. Este método se utiliza una vez por cada dato que se quiere obtener, puesto que como se verá en el desarrollo de la aplicación servidora, cada dato de un control es enviado en un flujo de datos diferente.

También pueden destacarse los métodos implementados para tratar las fechas y mostrarlas posteriormente en el display del teléfono o para emplearlas en la creación de la agenda. Sobre estos métodos, lo más destacable es el formato en el que se muestran los meses y que consiste en visualizar las tres primeras letras del nombre del mes en inglés. El motivo por el que se ha empleado este formato es que al obtener la fecha del terminal móvil, éste es el formato en el que devuelve el valor correspondiente al mes. Por sencillez en el desarrollo de la aplicación se ha mantenido este formato original tanto en su visualización, como en su empleo para almacenarlo en el `RecordStore`.

Sobre la conexión bluetooth, es importante comentar el atributo:

```
protected UUID uuid = new UUID(0x1101)
```

que es el identificador del perfil puerto serie (Serial Port Profile SPP).

También hay que destacar que en la conexión bluetooth se ha optado por no utilizar autenticación ni encriptación, caso identificado por el atributo:

```
protected int connectionOptions =
                                ServiceRecord.NOAUTHENTICATE_NOENCRYPT;
```

Tabla 2. Tabla de métodos de la clase `MainMIDlet`.

MÉTODO	PROPIO	PAQUETE	DESCRIPCIÓN
`public MainMIDlet()`	SÍ	-	Constructor.
`protected void startApp()`	NO	`javax. microedition. midlet`	Inicia la ejecución del MIDlet
`protected void pauseApp()`	NO	`javax. microedition. midlet`	Para la ejecución del MIDlet
`protected void destroyApp(boolean incondicional)`	NO	`javax. microedition. midlet`	Destruye el MIDlet
`public void commandAction(Command c, Displayable d)`	NO	`javax. microedition. lcdui`	Gestiona los comandos que pulsa el usuario y su funcionalidad
`private String getMonth(Date date)`	SÍ	-	Obtiene el mes a partir de una fecha para su posterior almacenamiento
`private String getDay(Date date)`	SÍ	-	Obtiene el día a partir de una fecha para su posterior almacenamiento
`private String getHour(Date date)`	SÍ	-	Obtiene la hora a partir de una fecha para su posterior almacenamiento
`private String getMinutes(Date date)`	SÍ	-	Obtiene los minutos a partir de una fecha para su posterior almacenamiento
`private String getYear(Date date)`	SÍ	-	Obtiene el año a partir de una fecha para su posterior almacenamiento
`private void startBTConnection()`	SÍ	-	Lanza la conexión bluetooth.
`private void startDeviceInquiry()`	SÍ	-	Obtiene un agente bluetooth y llama al método `startInquiry`
`public void deviceDiscovered(RemoteDevice btDevice, DeviceClass cod)`	NO	`javax. bluetooth`	Almacena los dispositivos bluetooth descubiertos
`public void inquiryCompleted(int discType)`	NO	`javax. bluetooth`	Método que se llama cuando acaba el descubrimiento de dispositivos. Crea la lista con los dispositivos descubiertos
`private void startServiceSearch(RemoteDevice device)`	SÍ	-	Obtiene el identificador bluetooth del dispositivo local.
`public void servicesDiscovered(int`	NO	`javax. bluetooth`	Método que se llama cuando uno o varios

transId, ServiceRecord[] records)			servicios son descubiertos
public void serviceSearchCompleted(int transID, int respCode)	NO	javax. bluetooth	Método que se llama cuando ha acabado la obtención de servicios
private void handleConnection(final String url)	SÍ	-	Controla la conexión bluetooth y el intercambio de datos.
private void makeInformationAreaGUI()	SÍ	-	Visualiza el Displayable Form con los datos obtenidos
private String getDeviceStr(RemoteDevice btDevice)throws Exception	SÍ	-	Obtiene el nombre del dispositivo bluetooth descubierto

Métodos de la clase `RegisterData`

En esta clase, encargada del almacenamiento persistente, se emplea un único `RecordStore` cuyo nombre es `data`.

Este `RecordStore` tendrá varios registros binarios o records en formato `String`, que contendrán toda la información asociada a un control. Es decir, cada control de glucosa se almacenará en un record, de modo que el total de registros almacenados en el `RecordStore` dependerá de la cantidad de mediciones que se haga el usuario; para este proyecto se ha estimado que un usuario se hará un máximo de siete controles de glucemia al día, con lo que se tendrían 2562 controles al año. La funcionalidad de almacenamiento está pensada para que se almacene como máximo esta cantidad, de modo que se puedan enviar por HTTP un máximo de 2562 controles. Esta función se ha implementado de manera que cuando haya almacenados los controles correspondientes a un mes y siguiendo la estimación anterior, en el terminal móvil aparecerá un mensaje al usuario advirtiendo que para almacenar el siguiente control, previamente debe eliminar todos los controles almacenados. Al usuario se le facilita la opción de enviar primero los controles por HTTP y después eliminarlos.

También es importante destacar que en los métodos que se emplean para obtener los datos del `RecordStore`, éstos se obtienen en formato `String` y se van añadiendo como opciones de un objeto `Displayable List`.

Como se puede ver en el apartado anterior, la clase `RegisterData` implementa el interfaz `RecordFilter`, que se usa para obtener los registros filtrados por mes y por día y así visualizarlos correctamente en la aplicación.

Por último, y aunque se hable de ello en el capítulo de "Pruebas", cabe comentar que el tamaño del `RecordStore` después de haber almacenado un control es de 93 Kilobits. Después de almacenar un segundo registro el tamaño pasa a ser de 182 Kilobits. La diferencia se puede deber a la extensión del campo "Comentario", ya que es el único campo con extensión variable. Teniendo en cuenta estas medidas, y estimando al alza una media de 100 Kilobits por cada registro, no parece que vaya a haber problemas de almacenamiento sobre todo si se tiene en cuenta que esta aplicación se ha probado después de instalarla en una tarjeta de memoria con una capacidad disponible para almacenar records de 2,9 GB.

Tabla 3. Tabla de métodos de la clase `RegisterData`.

MÉTODO	PROPIO	PAQUETE	DESCRIPCIÓN
`public boolean matches(byte[] candidate)throws IllegalArgumentException`	NO	`javax.microedition.rms`	Método del interfaz `RecordFilter` que filtra los resultados por mes y por día
`public RegisterData()`	SÍ	-	Constructor. Crea el `RecordStore`
`public void addData(String month, String day, String hour, String minutes, String year, String gluco, String comment)`	SÍ	-	Almacena los datos en el `RecordStore`
`private List printDataHelper(RecordEnumeration re)`	SÍ	-	Devuelve un `Displayable List` con la lista de los controles en formato visualizable al usuario a partir de los índices de los records
`public List printDays(String month, String day)`	SÍ	-	Devuelve los controles filtrados por mes y día
`public List printData()`	SÍ	-	Llama al método `printDataHelper`
`public int checkNumRecords()`	SÍ	-	Método que se encarga de comprobar el número de records almacenados en el `RecordStore`. Si este número es igual o superior que 2562, el usuario deberá eliminar los records y el `RecordStore` si quiere introducir más controles.
`private void close()`	SÍ	-	Cierra el `RecordStore`
`public void closeAndDelete()`	SÍ	-	Comprueba si el `RecordStore` está cerrado y lo borra.
`public void deleteControls()`	SÍ	-	Borra los records del `RecordStore`
`public int[] getMemoryAndRecordStoreSize()`	SÍ	-	Método de prueba que permite obtener el tamaño del `RecordStore` en bits y la memoria disponible en el teléfono también en bits.

Métodos de la clase `SendData`

Ésta es la clase encargada de enviar los datos por http. Para ello, se abre una conexión con la URL indicada y se van enviando los datos en formato `String`. Como los datos se envían a un servlet, la forma que tendrá la URL será la siguiente:

```
http://host:port/gluco/index?datos=+dato
```

Donde `host` será el nombre del equipo o su dirección IP, puerto será el número de puerto en el que esté escuchando el servidor web, `gluco` es en este caso el directorio donde está almacenada la aplicación web, `index` es el nombre que referencia al `servlet`, `datos` es el nombre del parámetro que se envía y `dato` es el nombre de la variable que representa el `String` que se envía.

Como los datos se envían como tipo: `MIME application/x-www-form-urlencoded` que se utiliza para pasar los datos a un formulario web, en la URL no pueden introducirse espacios. Como el `String` que se envía con el control contiene espacios que separan los datos, se ha creado el método `deleteSpaces` que sustituye los espacios por puntos ".". Posteriormente, en el servlet, se deshace este cambio. El motivo por el que se ha empleado un punto "." Para separar los datos es porque ha resultado el carácter más cómodo de escuchar por parte de una persona usuaria de un lector de pantalla.

Tabla 4. Tabla de métodos de la clase `SendData`.

MÉTODO	PROPIO	PAQUETE	DESCRIPCIÓN
`public SendData()`	SÍ	-	Constructor. Obtiene los datos almacenados y parsea la URL
`public void send()`	SÍ	-	Crea una conexión HTTP por cada dato que se quiere enviar y llama al método download
`private void download(String data)`	SÍ	-	Crea la conexión HTTP con la URL indicada
`private void deleteSpaces(String[] characters)`	SÍ	-	Elimina los espacios en la URL

3.3. Aplicación Servidora

En esta sección se explicará el funcionamiento de la aplicación servidora viendo la estructura de clases y de métodos que la componen, incidiendo en la obtención de los datos desde el medidor de glucosa y de su envío al teléfono móvil.

3.3.1. Descripción de la Aplicación Servidora

Para desarrollar esta aplicación se ha utilizado la librería Bluecove, que implementa el API JSR 82 para J2E. La aplicación servidora está pensada para ser ejecutada en un medidor de glucosa, de modo que la que se ha empleado en este prototipo no es más que una aplicación de consola que se ejecuta en un PC y que sirve como puente entre los datos obtenidos del medidor de glucosa y almacenados en el PC, y su posterior envío al teléfono móvil.

Hay que comentar una serie de consideraciones por las que no se ha creado una aplicación para que sea ejecutada en un medidor de glucosa:

- Hasta el momento de realización del proyecto, solamente había en el mercado un glucómetro con bluetooth integrado. Si bien podría haber sido ésta una opción viable, no lo fue finalmente debido a que el protocolo empleado en la transmisión bluetooth es propietario del laboratorio creador del medidor de glucosa.
- Este ha sido el motivo por el que el prototipo no puede probarse en un dispositivo real, y queda a la espera de que en un futuro sea viable.

3.3.2. Estructura de Clases y de Métodos

A continuación se muestra la relación de clases y de métodos de la aplicación servidora siguiendo la misma estructura que en el apartado anterior.

Relación de Clases

La aplicación servidora está compuesta por dos clases. La clase `BTServer` será la clase principal que contendrá el método `main`, y que por tanto será la clase a ejecutar. La otra clase, `ParserFile`, será la encargada de parsear los datos obtenidos del medidor de glucosa.

Tabla 5. Tabla de clases de la aplicación servidora.

CLASE	HEREDA DE	INTERFACES QUE IMPLEMENTA	DESCRIPCIÓN
`BTServer`	-	-	Clase principal. Crea el servidor bluetooth y envía los datos al dispositivo móvil
`ParserFile`	-	-	Parsea el fichero que contiene los controles de glucosa y obtiene la información que se envía al dispositivo móvil

Relación de Métodos

En este apartado se podrá ver una tabla por las dos clases que forma la aplicación servidora precedida por una explicación de los formatos empleados y que resulta más interesante comentar.

Métodos de la clase `BTServer`

Como se ha comentado, esta es la clase que contiene el método `main`. Cuando se ejecute en un terminal de consola, se le deberán pasar dos argumentos. El primero de ellos será el nombre del fichero del que tiene que obtener los controles de glucosa y el segundo será un entero que indicará el control que se quiere enviar al teléfono.

Si bien la aplicación servidora está pensada para que envíe solamente el último control almacenado, se ha implementado la opción de enviar (de uno en uno) más controles para que las pruebas resulten más llevaderas. De no haber hecho esto, la única manera de enviar más controles al teléfono móvil sería cambiando de fichero continuamente. Esta función, enviar diferentes controles de un mismo fichero, se utiliza escribiendo el segundo argumento en la ejecución de la aplicación, de manera que si el argumento es "0", el control que se enviará será el último, si es "1" se enviará el penúltimo, y así sucesivamente hasta un total de 21 controles. Por tanto, los datos correspondientes a un control se almacenarán en variables de tipo `String`, una por cada dato de un control, para posteriormente enviarlos usando un flujo saliente de datos (`DataOutputStream`) por cada una de ellas.

Tabla 6. Tabla de métodos de la clase `BTServer`.

MÉTODO	PROPIO	PAQUETE	DESCRIPCIÓN
`public BTServer()`	SÍ	-	Constructor. Obtiene el dispositivo bluetooth local, crea la URL para la conexión bluetooth y llama al método `serverLoop`
`private void serverLoop (StreamConnectionNotifier notifier)`	SÍ	-	Crea un bucle infinito en el que se esperan conexiones entrantes
`private void handleConnection(StreamConnection conn) throws IOException`	SÍ	-	Gestiona las conexiones entrantes y envía los datos
`Public static void main(String[] args)`	-	-	Lanza la ejecución.

Métodos de la clase `ParserFile`

Esta será la clase encargada de parsear el fichero que contiene los controles de glucemia. Su constructor recibirá como parámetros el nombre del fichero y un entero que indicará el control que se quiere obtener del fichero en cuestión; parámetros que se corresponden con los argumentos que hay que utilizar en la ejecución de la aplicación

servidora. El fichero de entrada es un documento con formato HTML, generado a partir de la transmisión de los datos desde el medidor de glucosa al PC. Para obtener cada dato, que en este caso serán el mes, el día, el año, la hora, los minutos y el nivel de glucosa, se ha optado por implementar el código en el propio constructor de la clase.

A este respecto cabe destacar el uso del método `split` ("separador"), el cual, pasándole el carácter de separación, devuelve un array cuyas posiciones contienen el dato. Por ejemplo, si se tiene la cadena de caracteres "`Jan.25.15:15.2010.125`". Empleando el método `split` de la siguiente manera: `Cadena.split(".")`, el método devolverá un array de Strings cuyas posiciones contendrán:

```
Jan.
25.
15:15.
2010.
125.
```

Este método se ha empleado debido a que el uso del método `tokenizer` está desaconsejado. También cabe destacar que este método no existe en el API de Java ME, por lo que se ha tenido que implementar un método que tenga una funcionalidad parecida.

A continuación se muestran unas líneas del documento HTML que se ha tenido que parsear para obtener los datos correspondientes a un control de glucosa.

```
"[…]
<td class="mN"> </td>
<td class="r"> </td>
</tr>
<!-- date record list will be created by firmware -->
<tr class="c">
<td class="o"> </td>
<td class="o">Miércoles</td>
<td class="l">10/14/2009</td>
<td>13:39</td>
<td class="mN">143</td>
<td class="mN"> </td>
<td class="mN"> </td>
<td><img src="img/bghi.gif"></td>
<td> </td>
<td> </td>
<td> </td>
<td> </td>
<td class="mN"> </td>
<td class="r"> </td>
</tr>
<tr>
<td class="o"> </td>
<td class="o">Miércoles</td>
<td class="l">10/14/2009</td>
<td>10:10</td>
<td class="mN">175</td>
<td class="mN"> </td>
<td class="mN"> </td>
<td><img src="img/bghi.gif"></td>
<td> </td>
<td> </td>
```

```
<td> </td>
<td> </td>
<td class="mN"> </td>
<td class="r"> </td>
</tr>
<tr class="c">
<td class="o"> </td>
<td class="o">Miércoles</td>
<td class="l">10/14/2009</td>
<td>07:57</td>
<td class="mN"> 93</td>
<td class="mN"> </td>
<td class="mN"> </td>
<td> </td>
<td> </td>
<td> </td>
<td> </td>
<td> </td>
<td class="mN"> </td>
<td class="r"> </td></tr>
[…]"
```

Tabla 7.Tabla de Métodos de la clase `ParserFile`.

MÉTODO	PROPIO	PAQUETE	DESCRIPCIÓN
`public ParserFile(String file, int numControl)`	SÍ	-	Constructor. Recibe el nombre del fichero por parámetro y un entero que indica el control que se quiere obtener de dicho fichero; parsea el fichero y obtiene todos los datos.
`private static int getArraySize(String file)`	SÍ	-	Obtiene el tamaño del array que contiene todas las líneas del fichero que se quiere parsear, para que se pueda inicializar correctamente.
`private static String[] readFile(String file)`	SÍ	-	Lee el fichero y almacena sus líneas en un array de Strings.
`public String getYear()`	SÍ	-	Obtiene el año.
`public String getMonth()`	SÍ	-	Obtiene el mes.
`public String getDay()`	SÍ	-	Obtiene el día.
`public String getHour()`	SÍ	-	Obtiene la hora.
`public String getMinutes()`	SÍ	-	Obtiene la hora.
`public String getGlucose()`	SÍ	-	Obtiene la glucosa.

3.4. Aplicación Web

En esta sección se verá cómo se reciben los datos enviados desde el teléfono móvil vía HTTP, y cómo se muestran en formato HTML, así como el motivo por el que no se ha utilizado una base de datos para almacenar los controles recibidos.

3.4.1. Descripción de la Aplicación Web

La aplicación web se basa en un servlet que recibe los datos enviados por el MIDlet y que los envía a una página JSP que es la que los muestra por pantalla. Esta aplicación está pensada para que sea usada por un centro médico. La aplicación es accesible, limitándose a mostrar los datos de forma tabulada en una tabla de contenidos. Los datos que verá el doctor serán los siguientes:

- Mes del control.
- Día.
- Hora.
- Año.
- Nivel de Glucosa.
- Comentario Asociado al Control.

Tal y como se acaba de comentar, la aplicación web está orientada a almacenar los datos que el paciente envía desde su terminal móvil para mostrarlos posteriormente al doctor que quiera consultarlos. Parece lógico por tanto pensar que la forma más apropiada para implementar esta funcionalidad es crear una base de datos a la que se accede desde el servlet para almacenar los controles. Posteriormente desde la página jsp se accederían a los métodos correspondientes para obtener los diferentes campos que conforman un control de glucosa completo.

Lo óptimo en este caso hubiera sido crear una base de datos para almacenar los controles recibidos desde el terminal móvil, y esa fue la intención inicial. Almacenar los datos en una base de datos para posteriormente mostrarlos en una página JSP y poder ordenarlos por fecha, por nivel de glucosa, por hora dada una fecha, etc. Sin embargo, la accesibilidad de los interfaces de desarrollo de bases de datos ha resultado ser una asignatura pendiente, y este ha sido el motivo por el que finalmente no se ha podido utilizar una base de datos. En el capítulo de “Pruebas” se comentará el intento de usar dos bases de datos y que finalmente no pudo ser.

En su lugar, se han creado variables globales e internas al servlet para almacenar los datos obtenidos de la URL de la siguiente manera: se ha creado un array de Strings por cada dato de un control (días, meses, horas, años, niveles de glucosa y comentarios). En las posiciones de estos arrays se almacenan los datos recibidos desde el terminal móvil y posteriormente se envían a la página JSP para su visualización.

Como la demostración de la aplicación trata de un prototipo que se centra en mostrar una pequeña funcionalidad, que en este caso es la visualización de los datos obtenidos por HTTP, y que formaría parte con total seguridad de una posible aplicación comercial, junto con otras funcionalidades que se detallarán mejor en el capítulo de “Ampliaciones”, ésta ha resultado una solución perfectamente válida.

3.4.2. Estructura de Clases y de Métodos

A continuación se muestra la estructura de clases y de métodos de la aplicación web siguiendo la misma estructura que en los apartados anteriores.

Relación de clases y ficheros

La aplicación web está formada por dos ficheros principales que son el servlet de nombre "`GlucoServlet`" y la página JSP llamada `index.jsp`.

Tabla 8. Tabla de Métodos de la clase ParserFile.

CLASE y FICHERO	HEREDA DE	INTERFACES QUE IMPLEMENTA	DESCRIPCIÓN
`glucoServlet`	`HttpServlet`	-	Servlet de la aplicación web que se encarga de obtener los datos, parsearlos y enviarlos a la página jsp
`index.jsp`	-	-	Página jsp principal en la que se muestran los datos en forma tabulada

Relación de Métodos

En este apartado se podrá ver una tabla por la clase que forma la aplicación web precedidas por una explicación de los formatos empleados y que resulta más interesante comentar.

Métodos de `GlucoServlet`

En esta clase es interesante comentar la implementación de sus dos métodos.

El método `parseData` recibe un String como parámetro. Este String será la cadena que se envía en la URL desde el terminal móvil. Tal y como se ha comentado en la sección de la aplicación móvil, para enviar los datos en la URL se tuvo que eliminar los espacios, sustituyéndolos por puntos ".". En este método, `parsedData`, se deshace el cambio hecho en el método `deleteSpaces` de la clase `SendData`, y se obtienen los datos de un control, actualizando las posiciones de arrays declarados como variables globales a la propia clase `glucoServlet`.

En el método `doGet`, se obtienen los datos y se incrementa un contador que marcará las posiciones del array en las que se van guardando los datos obtenidos de la URL.

Tabla 9. Tabla de Métodos de la clase ParserFile.

MÉTODO	PROPIO	PAQUETE	DESCRIPCIÓN
`public void doGet(HttpServletRequest request, HttpServletResponse response)throws IOException, ServletException`	NO	`javax.servlet.http`	Obtiene los datos enviados en la URL, los almacena en variables locales al servlet y los envía al jsp para su visualización
`private void parseData(String data)`	SÍ	-	Recibe como parámetro un control, lo parsea y obtiene los datos almacenándolos en variables locales al servlet

Capítulo 4 Pruebas

En este capítulo se mostrarán las pruebas que se han realizado en las diferentes aplicaciones que componen el prototipo, aplicación móvil, servidora y web. Estas pruebas pretenden medir la robustez de las aplicaciones y su comportamiento en situaciones concretas.

4.1. Pruebas de la Aplicación Móvil

Teniendo en cuenta las características del proyectando, deficiente visual, ha sido imprescindible contar con las pruebas realizadas en el terminal móvil puesto que la inaccesibilidad de los emuladores incorporados en los entornos de desarrollo existentes, hacían de ésta la única manera de ejecutar la aplicación y comprobar su funcionamiento. Esta forma de comprobar la ejecución de la aplicación móvil hacían necesarios los siguientes pasos:

1. Conexión del teléfono al PC vía USB.
2. Copia del fichero ".jar" en la tarjeta de memoria del teléfono.
3. Desconexión del teléfono del PC.
4. Instalación de la aplicación.
5. Comprobación de su funcionamiento

En ocasiones esta tarea ha resultado tediosa, sobre todo teniendo en cuenta que cuando se producía algún error en ejecución y éste no era capturado en la excepción correspondiente, la aplicación dejaba de funcionar en vez de salir de la aplicación. Por este motivo, y aunque a priori pueda parecer que no tiene relación con el presente proyecto, sería interesante abordar en un futuro la implementación de un emulador accesible para un lector de pantalla. No obstante, cabe destacar que en sucesivas pruebas, se comprobó que al conectar el teléfono móvil en modo "PC Suite" éste podía seguir conectado al PC mientras se reinstalaba la aplicación móvil reduciendo ligeramente el tiempo de ejecución de las pruebas.

A continuación se encuentran las pruebas realizadas en la aplicación móvil. En este caso, y a diferencia de las pruebas que se han hecho en las aplicaciones servidora y web, aparecen dos tablas; en la primera de ellas, se describen las pruebas unitarias realizadas a la aplicación móvil y que consisten en pruebas de funcionamiento de la aplicación. En la segunda tabla, se puede ver una relación de pruebas destinadas a comprobar la robustez de la aplicación durante la sucesión de diferentes situaciones que se han considerado habituales, como puede ser la entrada de una llamada mientras se hace uso de alguna de las opciones de la aplicación.

Es importante tener en cuenta que en la aplicación móvil las pruebas realizadas para el envío de datos por http se han hecho utilizando un teléfono móvil Nokia N95 que cuenta con tecnología wifi para el acceso inalámbrico a Internet.

Tabla 10. Tabla de pruebas Unitarias de la aplicación móvil.

PRUEBA	RESULTADO	COMPORTAMIENTO
Acceso a toda la información por parte del lector de pantalla.	SATISFACTORIO	El lector de pantalla accede a toda la información sin problemas.
Obtención de datos vía Bluetooth con el Bluetooth encendido.	SATISFACTORIO	Aparece inmediatamente la vista (creada en la aplicación) "Buscando Dispositivos Bluetooth".
Visualización de la lista de dispositivos bluetooth detectados.	SATISFACTORIO	Cuando la aplicación detecta dispositivos bluetooth activos, aparece un `Displayable List` que los muestra.
Almacenamiento de más controles de los permitidos por la aplicación.	SATISFACTORIO	La aplicación lanza un mensaje de aviso al usuario indicando que previamente tiene que eliminar los datos almacenados.
Visualización del número máximo de controles (2562).	ESPERABLE	El resultado es esperable puesto que cuando se acceda a la opción "Todos los controles Almacenados", estos controles se acaban visualizando pero el terminal tarda aproximadamente 50 segundos en mostrarlos.
`TextBox` de Instrucciones con saltos de línea.	SEMI-SATISFACTORIO	Cuando se accede por primera vez a la vista correspondiente en la que hay un `Displayable TextBox` con saltos de línea en su texto, el lector de pantalla lee toda la información de seguido, lo que es correcto. Cuando se vuelve a acceder a la información con las teclas del cursor, se puede leer la información línea a línea, pero la lectura del lector de pantalla es lenta y un poco pesada.
TextBox de Instrucciones sin saltos de línea.	SEMI-SATISFACTORIO	Tanto si se accede por primera vez a la vista como si se intenta acceder usando las teclas del cursor para leer la información, el lector de pantalla lee toda la información de seguido. Este comportamiento obliga a tener que escuchar toda la información para acceder a un dato concreto.
Comportamiento de la agenda en años bisiestos.	SATISFACTORIO	La agenda detecta correctamente los años bisiestos y realiza el cambio correspondiente en el mes de Febrero.
Visualización de los días del mes en curso.	SATISFACTORIO	La aplicación obtiene la fecha del sistema y muestra correctamente los días correspondientes al mes en

		curso.
Visualización de los días de un mes seleccionado.	SATISFACTORIO	La aplicación muestra correctamente los días del mes seleccionado.
Visualización de los controles de un día seleccionado dado un mes.	SATISFACTORIO	La aplicación muestra correctamente los controles correspondientes al día seleccionado en la agenda, filtrando por ese día y por el mes al que pertenece.
Ubicación de los botones de menú.	NO SATISFACTORIO	La ubicación de los botones varía según el modelo de teléfono empleado.
Almacenamiento de los controles.	SATISFACTORIO	Los controles se almacenan correctamente.
Eliminación de los records y del `RecordStore.`	SATISFACTORIO	Tanto los registros (*records*) como el `RecordStore` se eliminan correctamente.

Tabla 11.Tabla de pruebas Generales.

PRUEBA	RESULTADO	COMPORTAMIENTO
Obtención de datos vía Bluetooth con el Bluetooth apagado.	SATISFACTORIO	El teléfono móvil (no la aplicación) gestiona el encendido del dispositivo Bluetooth mediante un mensaje de consulta al usuario.
Detección de dispositivo Bluetooth cuando no hay ningún dispositivo activo.	NO SATISFACTORIO	La aplicación permanece buscando dispositivo permanentemente o hasta que el usuario decida finalizar la búsqueda.
Envío de datos vía HTTP.	SATISFACTORIO	El teléfono (no la aplicación) gestiona la activación del dispositivo wifi y la búsqueda de puntos de acceso.
Recepción de una llamada entrante con la aplicación en uso.	SATISFACTORIO	La llamada se puede contestar. Al colgar, se vuelve a la aplicación y el lector de pantalla lee en primer lugar el título de la visita en la que se había quedado.
Cambio de vista del display (vertical a horizontal por el acelerómetro).	SATISFACTORIO	Cuando se coloca el teléfono en posición horizontal la vista en el display cambia y las teclas del joystick para moverse por las opciones son ahora las que antes eran izquierda y derecha.

4.2. Pruebas de la Aplicación Servidora

Las pruebas de la aplicación servidora han resultado las más sencillas puesto que se trata de una aplicación típica de consola. Solamente han resultado complejas (sobre todo en el tiempo) las pruebas que requerían de la interacción con el terminal móvil. Sin embargo, cabe destacar que para desarrollar y por tanto para hacer las pruebas necesarias, se ha tenido que instalar la librería BlueCove. Ésta ha resultado una de las configuraciones más laboriosas en lo que a tiempo se refiere.

En la siguiente tabla se describen las pruebas realizadas a la aplicación servidora, pruebas que van desde la comprobación del correcto funcionamiento de la aplicación hasta el comportamiento de la misma en casos extremos como podría ser el intento de ejecución con el dispositivo bluetooth apagado.

Tabla 12. Tabla de pruebas de la aplicación servidora.

PRUEBA	RESULTADO	COMPORTAMIENTO
Ejecución de la aplicación con el dispositivo bluetooth apagado.	ESPERABLE	Aparece un error en la consola indicando que la aplicación no ha podido tener acceso a la pila bluetooth del PC. La aplicación sale después del error.
Ejecución de la aplicación con el dispositivo bluetooth integrado en el PC activado.	NO SATISFACTORIO	El comportamiento es el mismo que si el dispositivo bluetooth estuviese apagado
Ejecución de la aplicación utilizando un dispositivo bluetooth USB.	SATISFACTORIO	La aplicación mantiene un comportamiento adecuado. Para salir de la aplicación hay que pulsar `CTRL+C` y a continuación confirmar que se quiere salir.
Comportamiento después del envío de datos.	ESPERABLE	Después de enviar los datos la aplicación vuelve a esperar una conexión entrante. Para salir de la aplicación hay que pulsar `CTRL+C` y luego confirmar.
Ejecución de la aplicación pasando como segundo parámetro un número de control que no existe.	NO SATISFACTORIO	Salta una excepción de tipo `ArrayIndexOutOfBoundsException`.
Parseo del fichero de controles y obtención de los datos.	SATISFACTORIO	El parseo del fichero se realiza de forma correcta y se obtienen los datos necesarios para enviar al teléfono móvil

4.3. Pruebas de la Aplicación Web

Las pruebas de la aplicación web se han hecho con un servidor Apache tomcat 5.5 y con la versión 1.6 del JDK. En este caso, la carencia de un entorno de desarrollo ha derivado en la continua consulta del directorio de logs para comprobar los errores producidos en ejecución.

En este apartado también cabe destacar que se ha intentado utilizar dos bases de datos para el almacenamiento de los controles recibidos por http. Estas bases de datos son My SQL y Microsoft Office Access, y en ambos casos han existido problemas de interacción entre los interfaces de las bases de datos y el lector de pantalla utilizado en el PC, de modo que tuvo que desecharse esta opción de almacenar los controles en una base de datos.

En la tabla siguiente aparecen las pruebas realizadas a la aplicación web, y que se centran en el correcto funcionamiento de la aplicación.

Tabla 13. Tabla de pruebas de la aplicación servidora.

PRUEBA	RESULTADO	COMPORTAMIENTO
Primero se envían los datos y después se accede al navegador.	SATISFACTORIO	Cuando se accede a la vista aparecen todos los datos que se han enviado en una tabla
Primero se accede al navegador y después se envían los datos.	SATISFACTORIO	Cuando se accede al navegador no hay ningún dato en la tabla. Después de enviar los datos, se actualiza el navegador y aparecen correctamente.

4.4. Terminales Soportados

En esta sección se enumerarán las características de los terminales utilizados para ejecutar el prototipo. Se hablará de las características del teléfono móvil, del medidor de glucosa y del PC empleados.

Interesa comentar sobre la aplicación móvil que el hecho de emplear Java ME y RMS como sistema de almacenamiento de los datos hace que la aplicación móvil sea completamente portable, pero, no obstante, si se piensa en los terminales que un usuario ciego o deficiente visual puede emplear utilizando un lector de pantalla, hay que destacar que las limitaciones vendrían por la capacidad de dichos terminales para soportar un lector de pantalla, de modo que las pruebas de la aplicación móvil se han realizado en un teléfono móvil Nokia 6210 Navigator con sistema operativo Symbian 9.3. Actualmente los sistemas operativos para los que está disponible el lector de pantalla Mobile Speak son Symbian y Windows Mobile.

4.4.1. Terminal Móvil empleado

La elección de Java ME en la implementación de la aplicación móvil se debe a su portabilidad y a su sencillez. Teniendo esto en cuenta, la limitación de los dispositivos viene dada por que el dispositivo en el que se instale tenga desarrollado un lector de pantalla para el sistema operativo que emplea.

En el caso que nos ocupa, el teléfono es un Nokia 6210 Navigator que al contar con sistema operativo Symbian, dispone de diferentes lectores de pantalla que pueden ejecutarse en él. Las características de este terminal son las siguientes:

- Modelo: Nokia 6210 Navigator.
- Interfaz de Usuario: S60 versión 3.2 FP 2.
- Sistema operativo: Symbian OS 9.3. Procesador: ARM 11, 369 MHz.
- Memoria Interna: 130 MB.
- Tarjeta de Memoria: Micro SD 4 GB.

4.4.2. Medidor de Glucosa empleado

Con respecto al medidor de glucosa, no se ha tenido la posibilidad de acceder al único medidor que hasta la fecha incorpora bluetooth, de modo que aunque en este caso se haya empleado un medidor concreto, hoy en día cualquiera ofrece la posibilidad de transferir los datos a un ordenador. Las características de este medidor son las siguientes:

- Modelo: Accu-Check Compact Plus de Laboratorios Roche.
- Transmisión de Datos: Infrarrojos al dispositivo Smart Pix (dispositivo receptor de infrarrojos).
- Soporte en Audio: facilita los niveles de glucosa mediante pitidos.
- Medición: cartucho de 17 tiras reactivas.
- Cantidad de sangre necesaria: 1,5 µL.
- Tiempo de Resultado: 5 seg.

4.4.3. PC empleado

El PC empleado para ejecutar la aplicación servidora ha sido un ordenador portátil Toshiba. Las características de este PC son las siguientes:

- Modelo: Toshiba Tecra A6.
- Procesador: 1.8 Ghz.
- Memoria RAM: 1 GB.
- Sistema Operativo: Windows XP Service Pack 3.

Capítulo 5. Historia del Proyecto

Este proyecto es el resultado del intento de aportar algo de luz a las necesidades de las personas ciegas o deficientes visuales que a lo largo de varios años se han ido descubriendo empíricamente y que se han ido documentando tras comprobar con el paso de este tiempo que a pesar de los años, estas necesidades no se han cubierto.

Desde un principio se ha considerado como buena la idea de que un teléfono móvil es un elemento que forma parte del día a día hasta el punto de que prácticamente nadie puede imaginarse su rutina sin él. Esta idea ha sido clave a la hora de decidir el medio para desarrollar soluciones para un colectivo que depende en exceso de las tecnologías. En efecto, podría haberse pensado que el desarrollo de un nuevo aparato con una funcionalidad específica podría solucionar una necesidad concreta, y de hecho, así podría haberlo sido; pero sin embargo, se decidió por crear una aplicación móvil para que la solución vaya con el usuario como un elemento más de la rutina que un teléfono móvil marca en su vida.

A lo largo de 2009 los comienzos del proyecto se basaban en la idea de crear una aplicación móvil en Symbian, Sistema operativo del que se sabía ya que podía contar con aplicaciones accesibles para un lector de pantalla. Sin embargo, haciendo de la necesidad virtud, el tener que desarrollar unas prácticas en lenguaje Java ME e intentar que estas prácticas fuesen accesibles, fue el punto determinante que marcó la naturaleza del proyecto. Se decidió optar por Java ME como lenguaje de programación para el desarrollo de una aplicación móvil accesible, aunque aún no estaba definido qué aplicación sería la definitiva.

Una de las primeras ideas fue la de crear un lector y generador de códigos QR (códigos de barras bidimensionales) cuyo objetivo era el de etiquetar productos caseros o medicamentos con las etiquetas generadas desde el teléfono móvil, para posteriormente poder leerlas con el mismo terminal, haciendo que la información que apareciese en la pantalla pudiese ser leída por un lector de pantalla.

La experiencia con aplicaciones comerciales desarrolladas en Java ME no era positiva, puesto que todas ellas empleaban en algún momento el interfaz de usuario de Bajo Nivel, lo que las convertía en inaccesibles. No obstante, sí que se podía sacar de ellas algo interesante, y es que los menús que las formaban sí podían ser leídos por un lector de pantalla. La inaccesibilidad de estas opciones motivó a pensar en Symbian como lenguaje de desarrollo. No obstante, la documentación que se consultó hizo parecer poco viable la implementación de una aplicación de estas características por parte de una persona ciega. Esto se debió a que al consultar la norma ISO correspondiente a los códigos QR, toda ella estaba repleta de dibujos y diagramas necesarios para la implementación de un software de estas características.

Volviendo a Java ME, la primera intención era la de crear una aplicación de telemedicina que funcionase en un teléfono móvil y que se conectase con un medidor de glucosa vía bluetooth. Varias investigaciones después, se concluyó que no había en el mercado de forma accesible un glucómetro que transmitiese datos por bluetooth. Solamente había medidores de glucosa con infrarrojos o con conexión USB.

La siguiente intención fue la de obtener información acerca del API Java TV para crear una aplicación que reprodujese la Televisión Digital Terrestre (TDT) en un teléfono móvil y que pudiese acceder a la información de la EPG con un lector de pantalla. La documentación concluyó que no había teléfonos móviles que incorporasen el JSR 927.

Atendiendo a otra necesidad, se decidió por crear una aplicación móvil que simulase una estación meteorológica, conectándose también por bluetooth con un aparato de estas características del que obtendría la información y la mostraría en la pantalla. Los resultados de la investigación llevaron a concluir que al igual que con los medidores de glucosa, no había estaciones meteorológicas que incorporasen bluetooth para transmitir sus datos.

Al final, apareció un glucómetro nuevo de Laboratorios Roche que podía conectarse por infrarrojos con el PC y enviar los resultados de los controles generando un archivo en formato HTML. Fue entonces cuando se decidió volver a la idea original, crear una aplicación de telemedicina que visualizase los controles de glucosa en la pantalla, tomando una serie de decisiones de acuerdo con la situación que se planteaba. No había glucómetros con bluetooth, y en caso de existir el protocolo era propietario de los Laboratorios, de modo que se decidió por crear una aplicación servidora en un PC con vocación de integrarse en un medidor de glucosa y que se limitaría a obtener los datos del glucómetro y enviárselos al teléfono móvil. El teléfono por su parte, obtendría los datos, los mostraría en la pantalla, y finalmente los enviaría por HTTP a un servidor que se encargaría de mostrarlos en una página web.

Una vez se definió el objetivo del proyecto, se comenzó por desarrollar la aplicación móvil. Si bien el prototipo final se asemeja mucho a la idea original de la aplicación móvil, hay algunas diferencias con respecto a la intención que se tenía en un principio, el primer prototipo contaba con una opción dentro de la opción "Vista Diaria" llamada "Introducir Control" y que estaba pensada para que el usuario fuese introduciendo en el terminal los controles que se iba haciendo con el medidor de glucosa junto con un comentario asociado a los mismos.

Contemporáneamente a esta idea, se pensó que lo mejor era que la aplicación servidora mandase todos los controles de glucosa obtenidos del glucómetro. Esta idea hizo plantearse qué hacer con los controles replicados, puesto que se iba a dar la circunstancia de que en el terminal estarían almacenados tanto los controles introducidos a mano por el usuario como los controles recibidos desde la aplicación servidora. No resultó ser una cuestión trivial puesto que para discriminar entre controles, existía la necesidad de que tanto el terminal móvil como el glucómetro estuviesen sincronizados; algo que de tratarse de una aplicación comercial podría haberse solucionado, pero que en este caso, al ser un prototipo, no pareció una buena opción.

Existía otro problema añadido que era el de introducir comentarios en controles ya almacenados desde hace tiempo, con la consiguiente obligación para el usuario de acordarse o tener apuntado en otro lugar los comentarios asociados a cada control. Atendiendo a las cuestiones anteriores, finalmente se optó porque la aplicación hiciese las veces de diario de controles, almacenando siempre el último control obtenido del medidor de glucosa. De hecho, ésta es una práctica común entre las personas con diabetes, y que tienen que acudir a su endocrino con un librillo a modo de diario en el que tienen anotados todos los controles, la fecha, la hora y un comentario aclaratorio.

Una vez se tuvo claro la funcionalidad que la aplicación debía aportar al usuario, se comenzó a codificar dichas opciones comenzando por las instrucciones, siguiendo por el almacenamiento y visualización de los datos, y prosiguiendo con la aplicación bluetooth. Tras haber acabado la aplicación bluetooth, tanto en el terminal móvil que hace las veces de cliente como en el PC que hace de servidor, se codificaron las vistas del menú de usuario (Diaria y Mensual).

La última funcionalidad que se implementó fue la del envío de los datos por HTTP. Se comenzó por la codificación en la aplicación móvil y se siguió con la implementación del servlet que recibiría los datos, los transformaría, y los devolvería al JSP para que éste los mostrase. Esta última parte, a priori quizá la más sencilla, resultó complicada debido a una serie de problemas con la tecnología y el entorno de desarrollo que se iba a emplear.

Se pensó en un primer lugar en usar como entorno de desarrollo Netbeans 5.5 (las versiones posteriores dieron algún problema de accesibilidad) pero finalmente, por comodidad a la hora de editar el servlet y la página JSP, se decidió hacer uso de Apache Tomcat 5.5 como servidor web. Sobre estos problemas de accesibilidad no se puede asegurar de manera rigurosa a qué se deben, pero sí se puede hablar sobre el comportamiento del lector de pantalla JAWS cuando interactuaba con el entorno de desarrollo Netbeans. Las últimas versiones de Netbeans dieron problemas de accesibilidad puesto que a la hora de navegar entre los menús, había opciones que el lector de pantalla no leía. Si bien este problema no sucedía con la versión 5.5, la capacidad de proceso del PC en el que se ha desarrollado este proyecto hacía poco práctico el uso de este entorno de desarrollo junto con el lector de pantalla JAWS. En efecto, en numerosas ocasiones, el lector de pantalla perdía el foco (lugar de la pantalla donde está situado para leer), cuya única solución era minimizar Netbeans, cerrar el lector de pantalla, volver a ejecutarlo, y volver a maximizar el entorno de desarrollo. Quizá pueda parecer que esto era una cuestión menor, y en ocasiones no resultaba excesivamente molesto, pero sí ralentizaba en exceso el desarrollo de la aplicación, sobre todo cuando la pérdida del foco se producía en medio del uso de algún asistente para crear, por ejemplo, un archivo vinculado al proyecto. En estas ocasiones, el perder el foco implicaba volver a realizar todos los pasos desde un principio sin la seguridad de que se pudiese completar la operación.

Finalmente, el problema que motivó usar Apache Tomcat para desarrollar la aplicación web fue el que en numerosas ocasiones el lector de pantalla dejaba de responder lo que obligaba a reiniciar el PC.

Como ya se ha comentado, la intención fue la de crear una base de datos para almacenar los controles y sus campos, pero la accesibilidad a los interfaces de creación y manejo de bases de datos resultó ser una asignatura pendiente. Por este motivo se decidió almacenar los datos de los controles en variables locales al propio servlet. Finalmente se ha obtenido un prototipo completamente accesible en todas sus aplicaciones.

5.1. Planificación del Proyecto

La duración total de este proyecto ha sido aproximadamente de nueve meses teniendo en cuenta que ha habido un trabajo importante de investigación sobre

diferentes tecnologías y funcionalidades para desarrollarlo. A continuación se mostrarán las diferentes fases del proyecto así como su duración aproximada:

- **Documentación inicial**: ha sido la primera fase, basada en la idoneidad de usar Symbian o Java ME así como la investigación de diferentes funcionalidades y maneras de abordarlas. La duración aproximada fue de tres meses.
- **Análisis y diseño**: hay una pequeña parte, de unas tres semanas aproximadamente, dedicada exclusivamente al diseño del menú de usuario, almacenamiento y visualización y el análisis de sus implicaciones. La mayoría de este trabajo se ha hecho en paralelo a la implementación del prototipo, haciendo y deshaciendo lo que se había hecho en muchas ocasiones.
- **Programación de las aplicaciones**: ha sido la fase más larga en el tiempo pero también la que ha requerido del mayor esfuerzo. La duración aproximada ha sido de cuatro meses con una dedicación aproximada de entre veinticinco y treinta horas semanales.
- **Pruebas y depuración**: es una fase que se ha ido llevando a cabo en paralelo con la implementación de las aplicaciones, así que es complicado cuantificar en semanas su duración. No obstante, se dará un porcentaje aproximado del tiempo que ha ocupado con respecto a la duración total del proyecto. Aproximadamente habrá ocupado un 20% del tiempo dedicado a la implementación de las aplicaciones.
- **Desarrollo de la memoria**: dos meses.

Capítulo 6. Conclusiones y Trabajos Futuros

En este capítulo se abordarán las conclusiones a las que se han llegado después de todo el trabajo, de investigación y de implementación, que se ha venido realizando durante estos meses. También se propondrán una serie de mejoras que se pueden realizar al prototipo que se ha desarrollado, así como una serie de proyectos relacionados con el desarrollo de aplicaciones móviles accesibles y centradas en dar solución a situaciones cotidianas de personas ciegas o deficientes visuales.

6.1. Conclusiones

En este proyecto se ha implementado un prototipo de telemedicina basado en tres aplicaciones accesibles: aplicación móvil, aplicación servidora y aplicación web. El objetivo principal del proyecto ha sido el de desarrollar una aplicación móvil accesible basada en Java ME y en las posibilidades que su interfaz de Alto Nivel ofrece para acceder a la información por parte de un software lector de pantalla.

Se ha estudiado la viabilidad de utilizar tanto Symbian como Java ME para desarrollar aplicaciones móviles accesibles y se ha demostrado que se pueden crear aplicaciones para teléfonos móviles accesibles empleando Java ME y utilizando correctamente las herramientas que su interfaz de alto Nivel ofrece.

Se ha conseguido detectar un problema que afecta a las personas ciegas o deficientes visuales y diabéticas y aportar una solución sencilla y usable.

Se ha comprobado la idoneidad de usar unos elementos con respecto a otros; como por ejemplo, el uso de un `CheckBox` dentro de un `Displayable Form` acompañado de un elemento `TextField` como vista principal en vez de un `Displayable List`.

Se ha comprobado también que es más cómodo, en lo que a interacción del usuario se refiere, usar un `Displayable List` para mostrar la información en forma de lista en vez de usar un `Displayable Textbox`. El elemento `TextBox` se ha reservado para mostrar información estructurada en párrafos.

También se han superado las pruebas de robustez cuando la aplicación móvil interactúa con otros usos comunes de un teléfono móvil, como puede ser la recepción de llamadas.

En la realización de la memoria, se ha conseguido hacer llegar una necesidad que una persona con discapacidad puede llegar a tener, así como comprender conceptos básicos como son “qué es un lector de pantalla y un sintetizador de voz”, “qué es un medidor de glucosa”, y por qué es necesario que existan aplicaciones de estas características.

Sin embargo, hay un objetivo que no se ha podido cumplir, que es el de contar con una aplicación final totalmente integrada con un medidor de glucosa comercial, y se ha tenido que recurrir a una aplicación servidora instalada en un PC que hiciese las veces de pasarela entre el glucómetro y el teléfono móvil.

6.2. Ampliaciones

Esta sección se debe abordar teniendo en cuenta que el proyecto en su conjunto es un prototipo de una posible aplicación real de telemedicina. Por tanto, la ampliación que hubiese sido interesante abordar es la de obtener una aplicación final orientada a su distribución en el mercado. Para ello, se deben revisar o ampliar los siguientes puntos:

1. Ampliaciones de la aplicación Móvil

   Los aspectos de la aplicación móvil que pueden ser objeto de mejora son el interfaz de usuario, analizando la idoneidad de las funciones que realiza así como la incorporación de otras nuevas, como por ejemplo podría ser el poder visualizar de forma accesible en el teléfono móvil la información web asociada al paciente o la creación de una vista dedicada a la visualización de estadísticas derivadas de los controles almacenados: nivel máximo de glucosa, nivel mínimo, nivel medio, número de mediciones diarias…

   Otra ampliación que tendría sentido se podría enmarcar en el ámbito de la privacidad y seguridad en el envío de los datos, haciendo que éstos se transmitiesen de forma cifrada.

2. Ampliaciones de la aplicación servidora

   Ésta es la parte del prototipo que claramente está sujeta a la mejora más importante, que sería su integración en un dispositivo medidor de glucosa.

   Con respecto a la ampliación de sus funciones, podría pensarse en el envío de todos los controles almacenados en el "glucómetro" en vez del último control almacenado. En este caso, más que una ampliación de las funcionalidades de la aplicación servidora, puede hablarse de un cambio de filosofía en la función final de toda la aplicación.

3. Ampliaciones de la aplicación web

   En el prototipo implementado se ha pensado únicamente en la visualización de los datos recibidos desde el terminal móvil en una página JSP. Sin embargo, tendría sentido hablar de las siguientes mejoras:

   a) Almacenamiento de los controles en una base de datos.

   b) Autenticación del doctor para acceder al expediente del paciente utilizando su número de colegiado.

   c) Creación de diferentes vistas: estadísticas de los controles de forma gráfica en las que se podría ver una gráfica en la que apareciese la tendencia de glucosa del paciente así como sus niveles máximos y mínimos de glucosa. En esta vista podría implementarse también la función por medio de la cual el doctor, al hacer click en un nivel de glucosa concreto, se visualice el comentario asociado a dicho nivel junto con el resto de la información (fecha y hora).

6.3. Trabajos Futuros

Para comprender los trabajos futuros propuestos primero hay que entender uno de los objetivos del proyecto, que es la creación de una aplicación móvil accesible en

Java ME. Entendiendo esto, puede verse el sentido de las propuestas de trabajos futuros basadas todas ellas en la creación de aplicaciones accesibles para teléfonos móviles empleando diferentes API's de Java ME como son la JSR 234, la JSR 256, la JSR 257 y la JSR 927.

6.3.1. Trabajos con Advanced Multimedia Suplements API (JSR 234)

Algunos de los trabajos que podrían realizarse utilizando este API son los siguientes:

1. Reconocedor de Colores

   Este trabajo está basado en un antiguo proyecto fin de carrera en cuya documentación se ha basado esta propuesta de intento de mejora de la aplicación para reconocer colores. En la fecha en que este trabajo se implementó, no existía el JSR 234 de modo que el principal problema que se observó fue la imposibilidad de acceder al control del balance de blancos y al control del flash de la cámara por medio de los API's de Java ME disponibles hasta el momento.

   Estudiando la documentación y los ejemplos que hay para el JSR 234, una de las extensiones que aporta este API con respecto al control de la cámara de fotos del teléfono móvil es la posibilidad de tener control sobre el balance de blancos y sobre el manejo del flash, en caso de que el teléfono lo tuviese.

   Sería interesante estudiar la implementación de nuevo de una aplicación de reconocimiento de colores basándose en estas nuevas funcionalidades. Si bien en el mercado existen aplicaciones implementadas en Symbian y de pago, puede ser importante desarrollar una aplicación en Java ME más portable.

2. Magnificador de Pantalla

   Con el `ImageTransformControl` se puede estudiar la implementación de un magnificador de pantalla, ya que este control que aporta el JSR 234 permite modificar el tamaño de las fotografías.

   No parece viable la implementación de un magnificador de pantalla al uso, es decir, una aplicación que aumente el tamaño de todo lo que aparezca en la pantalla ya que este control está orientado a la transformación de imágenes. No obstante, puede estudiarse la creación de un software que magnifique fotografías o texto en formato fotografía.

3. Sintonizador de Radio

   En la actualidad, los teléfonos que disponen de radio y de la posibilidad de usar un lector de pantalla cuentan con sintonizadores de radio integrados cuya accesibilidad más o menos es aceptable. A pesar de esto, no parece que uno de los objetivos de los desarrolladores de estos sintonizadores sea su uso accesible. Por este motivo, resulta de interés desarrollar un sintonizador en Java ME usando la `Tuner Capability` y teniendo en cuenta la accesibilidad del Interfaz de Usuario de Alto Nivel.

4. Reproductor de Archivos de Audio

   En la actualidad el lector de pantalla por excelencia que se usa en terminales móviles es el Mobile Speak, desarrollado por la empresa Codefactory. Este software tiene asociadas una serie de aplicaciones entre las que se encuentra un reproductor de archivos de audio. El desarrollo del lector de pantalla y de sus aplicaciones está hecho en Symbian para teléfonos Nokia y también cuentan con lectores para Smart Phones, Blackberries, PDA's.

   El lector de pantalla es de pago, y también sus aplicaciones que deben adquirirse por separado. Puede resultar interesante por tanto el desarrollo de un reproductor de archivos de audio en Java ME usando la `Music Capability` y que, al igual que con el sintonizador de radio, se use la accesibilidad del Interfaz de usuario de Alto Nivel. Para resultar más concreto, la `Music Capability` se utilizaría para la reproducción en sí de los archivos de audio, pero también, para reducir el volumen del archivo que se esté reproduciendo en ese momento, en caso de que el usuario acceda a las opciones del menú. Esta funcionalidad persigue que cuando el usuario ciego o deficiente visual esté escuchando un archivo y quiera acceder a las opciones del reproductor, el volumen de la canción disminuya automáticamente para que se pueda escuchar con nitidez la voz del lector de pantalla.

6.3.2. Trabajos con Mobile Sensor API (JSR 256)

Éste es seguramente uno de los APIs estudiados más interesantes, debido a la cantidad de aplicaciones que podrían implementarse. Además, el hecho de que pueda emplearse con CDC 1.0, aporta la opción de desarrollar aplicaciones que no solamente estén destinadas a teléfonos móviles.

1. Medidor de glucosa

   Sería interesante estudiar la opción de incluir una aplicación que pudiese medir la glucosa en un teléfono móvil. No obstante, la principal y obvia limitación que tendría una aplicación como ésta es que necesita que el teléfono incluya el sensor en cuestión. No se puede olvidar que el JSR 256 es un API para el manejo de sensores, no para la creación de los mismos.

   No obstante, y según la documentación estudiada, este API está indicado tanto para manejar un sensor incluido en el dispositivo en el que esté corriendo la aplicación, como un dispositivo externo que pudiese comunicarse con un teléfono móvil vía bluetooth e incluso infrarrojos; este API por tanto permitiría la integración total del prototipo desarrollado para este proyecto, pudiendo emplear Java ME en el propio "glucómetro" para obtener los análisis y para su posterior envío a un terminal móvil.

2. Aplicaciones que hagan uso del acelerómetro

   Hoy en día hay muchas aplicaciones que empleen las funciones que aporta el acelerómetro. Sin embargo, es interesante hacer hincapié en el desarrollo de aplicaciones que empleen el Interfaz de Alto Nivel para poder visualizar de forma accesible los datos obtenidos de este sensor.

3. Estación meteorológica integrada

   Según se vayan incluyendo más sensores en dispositivos móviles, es razonable pensar que se cuente con sensores de temperatura, presión, humedad, etc. A este respecto se puede proponer la implementación de una aplicación accesible que simule las funciones de una estación meteorológica.

   Se debe tener en cuenta las limitaciones asociadas a un dispositivo móvil como que cuente con los sensores apropiados.

6.3.3. Trabajos con Contactless API (JSR 257)

Con este API el principal proyecto que se puede proponer es el de una aplicación móvil que lea códigos de barras o códigos BIDi empleando, como se comenta a lo largo de todo este proyecto el API de alto Nivel para que el software en cuestión sea accesible.

Con el desarrollo de un lector y generador accesible de "tags visuales" junto con una base de datos, una persona ciega o deficiente visual podría tener una aplicación que le sirviese por ejemplo para leer cajas de medicamentos, envases de comida, cajas de CD's o DVD's, etc…

6.3.4. Trabajos con Java API TV (JSR 927)

En el momento de estudio de este API la configuración de Java ME indicada es CDC, debido a los requisitos de memoria y de proceso que necesitaría una aplicación que implemente este API. Sería interesante por tanto ver si en un futuro algún terminal móvil implementa el JSR 927 y abordar entonces la posibilidad de crear una aplicación accesible para la reproducción de la Televisión Digital Terrestre.

Éste es un caso especialmente interesante puesto que se pueden abordar cuestiones de accesibilidad no solo en relación al interfaz de usuario, sino también a la posibilidad de acceder por medio de un lector de pantalla a la información asociada a la EPG.

Apéndice A. Presupuesto

El presupuesto se dividirá entre el coste de los equipos y los honorarios.

A.1. Coste de Material

El equipo que se ha utilizado para desarrollar este proyecto ha sido un ordenador portátil Toshiba Tecra A6, un teléfono móvil Nokia 6210 Navigator, un dispositivo bluetooth USB y un medidor de glucosa Accu-Check Compact Plus de Laboratorios Roche.

El software que se ha empleado está compuesto por los siguientes programas: Java(TM) 6 Update 17, Java Access Bridge, Java(TM) Platform, Micro Edition Software Development Kit 3.0, Java(TM) SE Development Kit 6 Update 16, Apache Tomkat 5.5, que son gratuitos; SmartPix de Laboratorios Roche incluido con el medidor de glucosa, Windows XP Profesional y los lectores de pantalla Mobile Speak y JAWS.

Tabla 14. Tabla de costes de material.

CONCEPTO	**COSTE**
Ordenador Portátil Toshiba Tecra A6	1200 €
Teléfono Móvil Nokia 6210 Navigator	119 €
Medidor de Glucosa Accu-check compact Plus	78 €
Dispositivo bluetooth USB	11€
Windows XP Profesional	123 €
Mobile Speak	117 €
JAWS	795 €
TOTAL	**2443 €**

A.2. Coste de Personal

La duración del proyecto ha sido de nueve meses. Durante este tiempo ha habido 193 días laborables, en los que se ha trabajado 5 horas diarias. El total de horas de trabajo que se han invertido en la realización del proyecto ha sido de 965.

Según los varemos orientativos del Colegio Oficial de Ingenieros Técnicos de Telecomunicación (COITT), con este total de horas se debe aplicar un coeficiente reductor de Ch = 0.45; teniendo en cuenta que este es el coeficiente reductor para el tramo que va desde las 720 hasta las 1080 horas. La hora normal de trabajo se cotiza a 62 euros. Por tanto, los honorarios de trabajo en el proyecto ascienden a:

Honorarios = 965 Ch * 62€/hora = 26.923,5€.

A estos honorarios hay que añadirles el trabajo de dirección del proyecto realizado por la tutora estimado en un 7% de los honorarios y el 16% de IVA. Por lo tanto, el presupuesto total en remuneraciones asciende a 33.115,91€ (ver Tabla 15).

Tabla 15. Tabla de honorarios.

Ejecución	26.923,5€
Dirección	1.884,65€
IVA (16%)	4.307,76€
TOTAL	**33.115,91€**

A.3. Coste total

El presupuesto total sumando el coste de los equipos y de los honorarios asciende a 35.558,91€ (ver Tabla 16).

Tabla 16. Tabla del coste total

Equipo	2.443€
Honorarios	33.115,91€
TOTAL	**35.558,91€**

Apéndice B. Manual de Instalación

En este anexo se realizará una breve guía para la instalación se los programas que se necesitan para el desarrollo y ejecución de las aplicaciones.

B.1. Instalación de Java(TM) 6 Update 17

Este programa permitirá desarrollar, compilar y ejecutar aplicaciones Java. En primer lugar habrá ir a la página web *java.sun.com* y descargar el fichero `jdk-1_5_0_17-windows-i586-p.exe`. Es un fichero autoejecutable cuya instalación lanza un *Install Shield Wizard* y en el que habrá que seleccionar la instalación típica.

B.2. Instalación de Java Access Bridge

Esta aplicación está desarrollada por Sun Microsystems y se emplea para dotar de accesibilidad (para lectores de pantalla) a las aplicaciones cuyo interfaz gráfico esté creado en Java.

En primer lugar habrá que ir a la página *java.sun.com/javase/technologies/accessibility/accessbridge/index.jsp*, y descargar el fichero `accessbridge-2_0_1.exe`. Es un fichero autoejecutable cuya instalación lanza un *Install Shield Wizard* y en el que se indicará las máquinas virtuales en las que se va a instalar *Java AccessBridge*.

Después de la instalación, hay que hacer los siguientes cambios:

1. Localizar la ruta de instalación de *Java AccessBridge* (comúnmente en `C:\Java AccessBridge`).
2. Dentro de esta carpeta se encuentran dos subcarpetas (`doc` y `installerFiles`) y varios ficheros. La carpeta que interesa es `installerFiles`.
3. Se deben seleccionar los siguientes ficheros: `access-bridge.jar, jaccess-1_2.jar, jaccess-1_3.jar, jaccess-1_4.jar.`
4. Localizar la carpeta `\lib\ext` en los directorios de cada máquina virtual instalada en el equipo.
5. Copiar los ficheros del paso 3 en la carpeta del paso 4.
6. Añadir la ruta de instalación de *Java AccessBridge* a la variable de entorno `PATH`.

Tras reiniciar, el equipo tendrá soporte de accesibilidad para las aplicaciones gráficas hechas en Java.

B.3. Instalación de Java(TM) Platform, Micro Edition Software Development Kit 3.0,

Es el entorno de desarrollo utilizado para crear la aplicación móvil. Hay que ir a la página *java.sun.com/javame/downloads/sdk30.jsp* y descargar el archivo `sun_java_me_sdk-3_0-win.exe`. Es un fichero autoejecutable cuya instalación lanza un *Install Shield Wizard* y en el que hay que dejar marcadas las opciones por defecto.

B.4. Instalación de Apache Tomcat

Apache Tomcat es el servidor web en el que se ejecutará la aplicación servidora. Habrá que ir a la página *tomcat.apache.org* y descargar el archivo de la última versión de Apache Tomcat. Es un fichero autoejecutable cuya instalación lanza un *Install Shield Wizard* y en el que el asistente solamente tiene que configurar el número de puerto en el que se quiere que se ejecute el servidor, el nombre de acceso como administrador y la contraseña para el acceso.

Bibliografía

[1] Java ME.
http://leo.ugr.es/J2ME/INTRO/intro_8b.htm (26-03-2010[1]).

[2] Arquitectura Java ME.
http://grasia.fdi.ucm.es/j2me/_J2METech/index.html (26-03-2010).

[3] Configuración CLDC.
http://grasia.fdi.ucm.es/j2me/_J2METech/CLDC.html (26-03-2010).

[4] Perfil MIDP.
http://grasia.fdi.ucm.es/j2me/_J2METech/MIDP.html (26-03-2010).

[5] MIDP 3.0.
http://java.sun.com/developer/technicalArticles/javame/midp3_enhance/ (26-03-2010).

[6] Paquetes Opcionales
http://grasia.fdi.ucm.es/j2me/_J2METech/OptionalPacks.html (26-03-2010).

[7] Mobile Service Architecture (MSA).
http://developers.sun.com/mobility/midp/articles/msaintro/ (26-03-2010).

[8] Record Management System (RMS).
http://www.it.uc3m.es/celeste/docencia/Java ME/tutoriales/midp1_0 (26-03-2010).

[9] Generic Connection Framework (GCF).
http://developers.sun.com/mobility/midp/articles/genericframework/ (08-04-2010).

[10] RFC 2616 - Hypertext Transfer Protocol - HTTP/1.1. 1999.

[11] Bluetooth.
http://es.wikipedia.org/wiki/Bluetooth (26-03-2010).

[12] Perfiles Bluetooth.
http://es.wikipedia.org/wiki/Perfil_Bluetooth (26-03-2010).

[13] Protocolos Bluetooth.
http://es.wikipedia.org/wiki/Protocolos_Bluetooth (26-03-2010).

[14] JSR 82 Bluetooth API
http://developers.sun.com/mobility/midp/articles/bluetooth2/ (26-03-2010).

[15] Librería Bluecove
http://bluecove.org/ (26-03-2010).

[16] Medidor de Glucosa.
http://www.surmedical.com/wiki/index.php?title=Medidor_de_Glucosa_(o_Glucometro) (26-03-2010).

[17] Síntesis de Voz.
http://es.wikipedia.org/wiki/S%C3%ADntesis_de_habla (8-4-2010).

[18] Telemedicina.
http://es.wikipedia.org/wiki/Telemedicina (26-03-2010).

[19] Aplicación de Telemedicina de Roche.
http://mundoasistencial.com/la-telemedicina-aporta-una-mejora-asistencial-en-el-abordaje-de-la-diabetes-gestacional-segun-un-estudio%E2%80%8F/ (26-03-2010).

[20] Teleconsulta Neumológica.
http://www.revistaesalud.com/index.php/revistaesalud/article/viewArticle/81/292 (26-03-2010).

[21] Telemedicina Cardiovascular.
http://www.blogseitb.com/masquepalabras/2010/02/18/telemedicina-el-futuro-hecho-presente/ (26-03-2010).

[22] Navarrete Pardo, Consolación. "Estudio sobre Prestaciones de Móviles Symbian OS con Cámara para el Desarrollo de Aplicaciones para Discapacitados Visuales". Proyecto Fin de Carrera. Tutora: Celeste Campo Vázquez. Escuela Politécnica Superior de la Universidad Carlos Tercero de Madrid, Departamento de Ingeniería Técnica de Telecomunicación Especialidad Telemática, 2006.

[23] Tierno Alvite, Jónatan. "Aplicaciones De Tratamiento de Imagen sobre Terminales Multimedia J2ME". Proyecto Fin de Carrera. Tutora: Celeste Campo Vázquez. Escuela Politécnica Superior de la Universidad Carlos Tercero de Madrid, Departamento de Ingeniería Técnica de Telecomunicación Especialidad Telemática, 2004.

[25] Castro Fernández, Raúl. "Desarrollo de Software de Videovigilancia para sistemas Embarcados Distribuidos con ICE". Proyecto Fin de Carrera. Tutora: Marisol García Valls. Escuela Politécnica Superior de la Universidad Carlos Tercero de Madrid, Departamento de Ingeniería Técnica de Telecomunicación Especialidad Telemática, 2009.

[26] Florina Almenárez Mendoza. "Programación en Mobile Information Device Profile (MIDP) [pdf]. 2009.

[27] Motorola, Inc. "Mobile Information Device Profile for Java TM Micro Edition Version 3.0" [en línea: http://opensource.motorola.com/sf/projects/jsr271] (3-11-2009).

[28] Sun Microsystems, Inc. "Mobile Service Architecture" [pdf]. 2007.

[29] Nokia Corporation. "PC Connectivity Over Bluetooth in Java TM Applications Version 1.0" [pdf]. (8-12-2006).

Bibliografía Complementaria

http://www.linecity.de/INFOTECH_ACS_SS04/acs4_top_1.pdf (26-03-2010).

http://www.ifi.uzh.ch/~riedl/lectures/Java2001-Java ME.pdf (26-03-2010).

http://www.forum.nokia.com (26-03-2010).

http://java.sun.com/javame/technology/msa/jsr234.jsp (26-03-2010).

http://developer.sonyericsson.com/site/global/techsupport/tipstrickscode/java/p_jsr234_ (26-03-2010).

http://javiercancela.com/tag/jsr-256/ (26-03-2010).

http://developer.sonyericsson.com/community/docs/DOC-1995 (26-03-2010).

http://209.85.229.132/search?q=cache:kWXg0avta7UJ:www.morelab.deusto.es/images/talks/NFC.ppt+jsr+257&cd=4&hl=es&ct=clnk&gl=es (26-03-2010).

http://java.sun.com/developer/technicalArticles/javame/nfc_bluetooth/index.html (26-03-2010).

http://java.sun.com/developer/technicalArticles/javame/nfc/ (26-03-2010).

http://wiki.forum.nokia.com/index.php/CDC (26-03-2010).

http://jcp.org/en/jsr/tech?listBy=1&listByType=platform (26-03-2010).

http://wiki.forum.nokia.com/index.php/APIs_adicionales_en_Java_ME (26-03-2010).

http://wiki.forum.nokia.com/index.php/Java_ME_API_support_on_Nokia_devices (26-03-2010).

http://www.cesnavarra.net/cesdigital/Lists/Noticias%20CESDigital/DispFormCES.aspx?Li (26-03-2010).

http://www.mhproject.org/index.php/mhproject.php/2006/11/10/ique_es_una_xlet_ipara_que_sirven (26-03-2010).

http://www.interactivetvweb.org/tutorials/ocap/your_first_xlet (26-03-2010).

[1] Fechas de último acceso.